Consistency Maintenance Algorithms for Multiplayer Online Digital Games

CONSISTENCY MAINTENANCE ALGORITHMS FOR MULTIPLAYER ONLINE DIGITAL GAMES

By

Kyung Seob Moon

A DISSERTATION SUBMITTED IN FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
THE SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY
GRIFFITH UNIVERSITY
AUSTRALIA
MARCH 2007

Acknowledgements

"The reasonable man adapts himself to the world; the unreasonable one persists in trying to adapt the world to himself. Therefore all progress depends on the unreasonable man."

- George Bernard Shaw

First of all, I would like to express my sincere appreciation of Dr. Anne Nguyen, my Principal Supervisor. Without her unceasing efforts and encouragement, this thesis would never have been completed. She showed me endless support and has always been forgiving, under any circumstance. As an Associate Supervisor, Dr. Michael Blumenstein kindly supported and encouraged my research. Thank you very much for your kindness and help, Michael!

I would also like to humbly thank Amol Wagholikar, Hisham Alsaghier, Kylie Shoobridge, Layla Soroush, Vicky Wheeler, Prof. Yew-Chaye Loo, Yullian Fan, and everyone who encouraged me while I was staying in a deep corner of "the dungeon" trying to complete my PhD research. Special thanks must go to Prof. Hyung-Soo Kim and my fellow PhD Candidate, Shaun Xiang. Prof. Hyung-Soo Kim showed me endless support and gave me advice on not only academic but also important, general-life issues, while Shaun provided me with a pure, invaluable friendship which I shall never forget. I also wish to thank the School of ICT, Griffith University and the IT technical support team for providing excellent research facilities and support. Thanks are also due to Professor Tom Nguyen for his kindness, support, and erudite advice.

I have learnt a lot during the course of my PhD studies -- I have learnt not only academic matters but also (I would like to think) how to live more wisely. During this invaluable lesson, my wife was standing by me all the time. I do not believe that I can repay her love within this mere lifetime. Ji-Yeoun, I love you and I always will.

Last but certainly not least, I would like to thank my family and friends in South Korea. While I was doing my PhD studies, my father passed away. I sincerely dedicate this thesis to my father who is now in another world, and my mother who loves me so much and has truly shown me how to live correctly. Father and mother, I love and respect you very much!

K.S.M.

Statement of Originality

Date: **March 2007**

Author: **Kyung Seob Moon**

Title: **Consistency Maintenance Algorithms for Multiplayer Online Digital Games**

Department: **School of Information and Communication Technology**

Degree: **Ph.D.**

Year: **2007**

I declare this work has not previously been submitted for a degree or diploma in any university. To the best of my knowledge and belief, this thesis contains no material previously published or written by another person except where due acknowledgement is made in the thesis itself.

Signature: _____

Date: _____

Abstract

Multiplayer Online Digital Games (MODIGs) are gaining in popularity because of the strategic sophistication added when games are played against other humans, as opposed to computer artificial intelligence (AI) opponents. However, the actualisation of multiplayer games is not easy, due to their complexity. Multiplayer games are the combined applications of various areas, such as networking, graphics, AI, sound, and process optimisation. Among them, problems related to networking -- such as limitations in data transfer rate, latency, and jitter -- are the most difficult to resolve. Network latency cannot be avoided completely and introduces various problems such as inconsistency of player status, recognisable responsiveness, and irregular network lag.

Generally, the network architectures of MODIGs can be categorized into three groups: Client-Server (C/S), Peer-to-Peer (P2P), and hybrid. In general, MODIG designers prefer the C/S network architecture to the P2P system. The main reason for this is that the C/S model enjoys certain advantages, such as simplicity of consistency maintenance, improved security, efficient authentification, and ease of billing system management. However, the C/S architecture can cause network latency and often servers do become network bottlenecks. To solve this problem, server clustering methods are used, but these solutions may not be cost effective. This is why a number of games use the P2P network architecture, but in this structure the total number of players in any one game session is often limited, because of the network's bandwidth

constraints. In addition, the consistency maintenance issue becomes critical within this architecture.

There are two main approaches for maintaining consistency in MODIGs: conservative and optimistic. The former approach involves a send-and-wait philosophy, requiring acknowledgement frames and resulting in packet transfer delay, such that players may experience network latency. In the latter approach, the processes do not wait for other players' packets and advance to their own frames, thus no network latency occurs. However, when there is inconsistency between players, the processes must roll back to correct mis-ordered operations due to packet transfer delay. These can cause irritation and confusion to players, and thus the quality of game deteriorates. Overall, the optimistic approaches may not be suitable for network games.

To alleviate network latency and reduce bandwidth requirements in conservative consistency maintenance algorithms and the P2P-based approaches, a new system is proposed and designed. To reduce network latency, a conservative consistency maintenance algorithm named Locked Bucket Synchronisation (LBS) is proposed to mask latency and maintain perfect consistency among players. In addition, a distributed network architecture is adopted and a tree-based P2P system is proposed, to remove additional packet transfer delays between server and client. To reduce network latency caused by packet drop and delay, a smart transmission scheme (STS) is proposed. To alleviate the bandwidth problem, a packet aggregation method is introduced. These approaches are thoroughly examined and analysed.

To evaluate the efficiency of the proposed system, a network simulator, COMP2P, and a real network game, Duel-X, are implemented. The efficiency of the proposed consistency algorithm, LBS is compared with that of other approaches such as

Lockstep (LS) and Frequent State Regeneration (FSR). The system architectures of COMP2P and Duel-X as well as the results of experiments are fully documented.

The experimental results from COMP2P show that the proposed LBS algorithm outperforms the LS algorithm under all tested circumstances, in terms of game execution speed. The LBS algorithm with the tree-based P2P system achieves almost optimal frame rate under the condition of a 10% packet drop rate, while the LS algorithm with the tree-base P2P system performs at approximately 40% of optimal frame rate.

The efficiency of the Smart Transmission Scheme (STS) with the LBS algorithm is compared with the Blind Transmission Scheme (BTS) and the experimental results indicate that the BTS increases the bandwidth requirement of players by 100% while the STS raises by only 10% the optimal value of the bandwidth requirement without degrading game execution speed significantly.

Finally, the proposed packet aggregation method together with the LBS algorithm reduce by 50% the bandwidth requirement of players, without lowering game execution speed, when compared with the case of using the LBS algorithm only.

To verify and validate the efficiency of the LBS algorithm, various experiments are performed on Duel-X. The experimental results show that the LBS algorithm masks network latency without harming consistency between players. When the LBS algorithm is adopted, the rate of frame interval approaches the optimal values that can be achieved under the FSR algorithm. This implies that the LBS algorithm improves the playability of MODIGs without degrading consistency between players.

The overall experimental results show that actualisation of reliable and robust MODIGs is achievable with a combination of P2P-based architecture and conservative consistency maintenance algorithms.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
BS	Bucket Synchronisation
BTS	Blind Transmission Scheme
C/S	Client/Server
COMP2P	Compact P2P Network Simulator
CVE	Collaborative Virtual Environment
DARPA	Defense Advanced Research Projects Agency
DIS	Distributed Interactive Simulation
DR	Dead Reckoning
fps	frame per second
FPS	First Person Shooter
FSR	Frequent State Regeneration
GVT	Global Virtual Time
IPX	Internetwork Packet Exchange
LP	Logical Process
LS	Lockstep

LBS	Locked Bucket Synchronisation
MAT	Must-Arrive Time
MMOG	Massively Multiplayer Online Game
MMORPG	Massively Multiplayer Online Role Playing Game
MODIG	Multiplayer Online Digital Game
NES	Nintendo Entertainment System
NPAT	Next Packet Arrival Time
NSFNET	National Science Foundation Network
NVE	Networked Virtual Environment
OS	Operating System
P2P	Peer-to-Peer
PHBDR	Position History-Based Dead Reckoning
PPS	Packet per Second
RTP	Real-Time Application Level Protocol
RTS	Real-Time Strategy
RTT	Round Trip Time
SIMNET	Simulator Networking
STL	Standard Template Library

STS	Smart Transmission Scheme
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WOW	World of Warcraft

1. Introduction

1.1. Background

Multiplayer Online Digital Games (MODIGs) are computer/video games that can be played by multiple players at the same time. Players can be teamed up to achieve the same goals or compete against each other. The popularity of MODIGs is significant and continues to rise, mainly due to the increased strategic complexity of game play against other human being, as opposed to computer artificial intelligence (AI).

The actualisation of MODIGs, as opposed to single player or non-networked games, requires a higher level of programming knowledge in various areas such as networking, graphics, sound, AI, and process optimisation. It is not only the complexity of MODIGs that makes the actualisation a difficult task, but there are also various constraints such as processing power, bandwidth, latency, and jitter.

The development of computer hardware and networking technology enables the spread of high-end personal computers and broadband Internet connections, which are now common in developed countries. For this reason, the processing power and bandwidth problems have become less significant than before. However, due to the physical properties of data transmission, the network latency remains a major problem (Moon et al., 2006a).

Consistency is another important factor in MODIGs for state updating and player action fairness. Event ordering is one of the methods for maintaining consistency in traditional multi-process environments (Moon et al., 2005). Network latency and consistency maintenance are mutually exclusive. We will, therefore, examine various

consistency maintenance algorithms and suggest several approaches which can reduce network latency without significantly increasing bandwidth requirements.

1.2. Significance of the Research

The estimated value of the worldwide gaming market was over $30 billion in 2002 and has been growing rapidly. This, in fact, exceeded the size of the film industry (Shuster, 2003). Heng (2005) also reported that in 2004 online gaming subscription revenue was estimated more than US $1.09 billion in the Asia/Pacific region (excluding Japan), roughly 30% more than the previous year. This figure is expected to double by 2009.

The studies of real-time interaction between multiple users have focused not only on entertainment but also on other areas such as military simulation, business, and education. Collaborative Virtual Environments (CVEs) and Networked Virtual Environments (NVEs) are software systems which enable real-time interaction through communications on networks (Singhal, 1999) and have attracted many research scholars and developers. Reducing network latency and bandwidth requirements while maintaining consistency is one of main aims of the research. Therefore, several approaches for pursuing this main aim are presented in this thesis.

Creating reliable and robust military simulators that can support as many participants as possible has become a major aim of research in the military simulation area; various architectures and systems have been proposed and implemented (Calvin et al., 1995; Cohen, 1994; DIS, 1993; Macedonia et al., 1994; Miller et al., 1995; Pope, 1989; STOW, 1998). Massively Multiplayer Online Games (MMOGs) have a similar aim of supporting as many players as possible without affecting system stability or

increasing the financial burden. To achieve this aim, many resource management algorithms and approaches have been proposed (Abrams et al., 1998; Barrus et al., 1996; Greenhalgh & Benford, 1997; Morse, 1996). For the resource management problem, this thesis examines a packet aggregation method.

1.3. Research Problems

The major problems in the actualisation of current MODIGs are (Jiang et al., 2005):

- Network latency
- Bandwidth
- Inconsistency
- Responsiveness

Generally, network latency is caused by not only physical distances between players but also other aspects such as hardware failure and network congestion. Bandwidth limitations can be addressed by utilizing broadband technology which is widely available in most developed countries. However, even with broadband technology, inefficient network structure may cause bandwidth problems when the number of players increases. Furthermore, the development of multimedia and networking technology has enabled audio/video transfer which increases bandwidth requirements.

Most of the MMOGs have adopted the C/S network architecture because of the simplicity of consistency maintenance, better security and authentification, and the easy billing system. However, C/S network architectures introduce additional network latency compared with the P2P system.

For this reason, some casual network games prefer P2P system, but this architecture has a major problem related to network latency. In P2P systems, each node manages its own status. Therefore, inconsistency is not avoidable due to network delay. In addition, each node is required to transmit same packets to other nodes when certain transmit schemes (e.g. unicast) are applied. This may restrict the total number of players in a game.

Maintaining consistency between players (or users) has become a major issue in networked virtual environments. Structures of P2P network tend to suffer from this problem due to the lack of a central maintenance mechanism. Responsiveness is referred to as the time taken to display the result of a player's command. When responsiveness problems are noticeable to players, this may cause irritation, confusion and difficulty in game control.

1.4. Research Aims

The main aim of this research is to create efficient P2P-based MODIGs which minimize network latency, bandwidth, and inconsistency problems. To realise this aim, several sub goals are established as follows:

- Exploiting the advantages of tree-based P2P systems to reduce network latency and bandwidth requirements.

- Using the optimal packet re-sending frequency to alleviate network latency problem in case of packet drop. The packet drop includes not only undelivered packets but also the corruption of packets: the unwitting or deliberate alteration of the packet data during transmission process.

- Applying packet aggregation for the reduction of bandwidth requirements.
- Analysing the efficiency of the proposed P2P systems by comparing them with other P2P systems.
- Verifying and evaluating the efficiency of the proposed consistency maintenance algorithms.
- Utilising imperfect human vision to maintain consistency by adopting audio/video packet buffering concepts.

1.5. Organisation of the Thesis

Seven chapters in total are presented in this thesis. Chapter 2 consists of several literature review sections which include an examination of multiplayer online digital games, networking, concurrency control, resource management, existing problems and approaches.

Chapter 3 describes the research methodology and presents a system model used in this research and several modules: a communication module (tree-based P2P system), concurrency control module (locked bucket synchronisation algorithm), and a resource management module (packet aggregation and retransmission).

The experimental systems are examined and the implementation details of network simulator and a real-time multiplayer game are presented in Chapter 4.

Chapter 5 presents the full explanation of three consistency maintenance algorithms: Lockstep (LS), Locked Bucket Synchronisation (LBS), and Frequent State Regeneration (FSR). For evaluation, these algorithms are implemented and the

experimental results from two systems are analysed. To reduce network latency problems, a number of packet transmission schemes and a tree-based P2P system are also proposed in Chapter 6. The packet aggregation method for reducing the bandwidth requirement of each player is described in this chapter. Furthermore, various experimental results based on graph and tree-based P2P systems, consistency maintenance algorithms, and packet transmission schemes are presented in the chapter.

Chapter 7 provides a short summary of the research and an evaluation of the proposed systems. Avenues for future research are discussed at the end.

2. Literature Review

This chapter presents an extensive literature review relating to Multiplayer Online Digital Games (MODIGs). The reviewed literature covers various sections such as definition and history of games, networking issues in MODIGs, consistency maintenance, resource management, existing problems and approaches.

2.1. Multiplayer Online Digital Games

The following sections present the definition of digital games and the significance of multiplayer online digital games. A brief history of digital games and multiplayer online digital games are described, followed by an outline of the current challenges in multiplayer online digital games.

2.1.1. Overview

According to Salen and Zimmerman's definition, a game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome (Salen & Zimmerman, 2003). In games, the system provides contexts for interaction and the system may be mechanical and/or mathematical. Players are the main characters in the game who participate in the virtual environment to compete with other players or AI-controlled opponents to achieve certain goals. The goals can be pre-defined by the system or by the players themselves. The rules of games limit the degree of freedom in games. In other words, the rules define what players can do and can not do. When the main and/or sub goals are completed, the numerical scores of players' achievements are displayed.

The term ‘digital games’ which appeared in the online academic journal, Game Studies, combines two conventional terms: video and computer games. Earlier, the term ‘video games’ was used to refer to games on arcade machines, consoles and PCs, by Herz (1997), Poole(2000) and Wolf(2001). Burnham (2001) delineates the definition of video games as an electronic game played by manipulating images on a video screen. This is a narrow meaning of digital games which utilises a video display such as a monitor or television because not all digital games employ only video devices.

Even though the development of audio games for the blind has started but their usage is broadened by sound artists, game accessibility researchers, mobile game developers and video gamers. Warp, one of the most famous console game development company in Japan, commercialised an audio game called ‘Real Sound: Kaze no Regret (The regret of wind)’ in 1999 (Wikipedia: Audio Game, 2006). The term ‘digital games’ also includes various forms of game platforms such as personal computers, TV-attached game consoles, hand-held devices, mobile phones, and so on.

The shareware computer game Doom was one of the most popular games in computer game history and 15 million copies have been downloaded around the world. The 3D environment and the various modes of networking play such as co-operative and death-match modes have been cited as factors contributing to the game’s popularity (Doom World, 2003). Since the great success of this game, the majority of games feature network play functionality, and network games have become more and more popular because of the increased strategic complexity of network play (Smed et al., 2001).

The Multiplayer Online Digital Games (MODIGs) can be categorised into two groups, namely the Massively Multiplayer Online Games (MMOG) and casual games. The main differences between them are:

- The size of players participating in concurrent time and space;
- The persistency of environment; (The persistency denotes that the virtual world will evolve disregarding the participation of the user. This characteristic of MMOG may cause the addiction of the games.) and
- Network architecture.

These two groups can be further categorised by their game genres, and will be further explained in detail in following sub sections.

The significance of MODIGs is reflected in both business and research sides. From the business point of view, the market size of gaming industry and its online gaming subscription revenue are revealed, that the revenue of the worldwide gaming market was over $30 billion in 2002 exceeded that of movie industry (Shuster, 2003) and this figure has been growing rapidly. Heng (2005) also reported that in 2004 online gaming subscription revenue was estimated to be more than US $1.09 billion in the Asia/Pacific region (excluding Japan), roughly 30% more than in 2003. This figure is expected to double by 2009.

Digital games have been used not only just for entertainment but also for various purposes such as education, research, military training, etc. The Guardian's articles, 'Let the games begin' (Dodson, 2004) and 'Welcome to play school' (McClean, 2004) report how digital games are adopted as formal educational contexts in the U.K. 'HistoryCity' is a virtual community which was developed by Das et al. (1997) for

children aged 7-11 for educational purposes. The 'HistoryCity' utilises a network architecture called NetEffect, which is similar to what currently being used by MMOG server-clustering systems.

Consistency maintenance and interest management algorithms have received a great deal of research focus and various approaches and algorithms have been proposed. The former is targeted by researchers who study various algorithms for optimising the efficiency of consistency maintenance mechanisms in Collaborative Virtual Environment (CVE) and Networked Virtual Environment (NVE). The latter was initially funded and researched by US military departments for supporting as many concurrent participants as possible for military simulation (DIS, 1993; Macedonia et al., 1994). Currently, these areas have drawn attention from researchers and developers of MMOGs on increasing the number of players that game servers can handle and decreasing the bandwidth requirement of each player.

2.1.2. History

It is arguable to define the first appearance of digital games. It is ambiguous due to the characteristics of computer and/or video games at the initial state. The arguable first well-known computer/video game is a missile simulation which was created by Thomas T. Goldsmith Jr. and Estle Ray Mann in 1947 (Burnham, 2001). However, this game can not be called a digital game because the employment of an analogue computer system utilised amplifiers, vacuum tubes, and oscilloscopes. In 1952, A. S. Doublas implemented a tic-tac-toe game named 'Noughts and Crosses' on a EDSAC (Electronic Delay Storage Automatic Calculator). Strictly speaking, this is not regarded as a digital game due to the aforementioned reason (Kerr, 2006).

Wolf (2001) mentioned in his book that the arguable first digital game is 'Spacewar' developed by Steven Russell in 1962. Figure 2.1 shows the image of the game. Later, Ralph Baer, known as 'Father of Video Games' released the first console, named 'Magnavox Odyssey' in 1972. Meanwhile, the new improvement of home consoles appeared. Atari debuted on game market in 1972 with its first game 'Pong'. Nintendo, one of the current major console game companies in Japan, released its own version of the ping-pong game in 1977, and Apple II which equipped with colour graphics and game paddles was also released in the same year (Burnham, 2001).

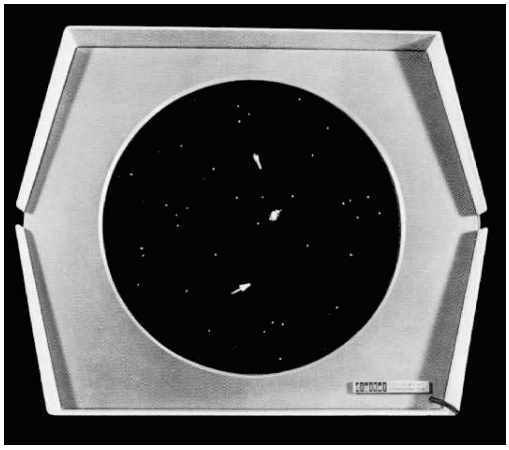


Figure 2.1: Spacewar (Burnham, 2001)

As the technology of computer hardware and software has advanced significantly, the quality of games is also improved. The converse of the previous proposition might be true. The pursuit of more sensational quality of images, sound, logics, and system in games have pushed the fast development of the advanced computer technology, both in hardware and software. The debut of new console game platforms such as SEGA Saturn, Nintendo 64, Sony PlayStation 1, 2, 3, PSP (PlayStation Portable), MS X-Box and X-Box 360, etc in console game market and the appearance of hardware devices such as 3D video cards, sound cards, multi-core CPU, game related interface devices

such as joystick, head mounted display (HMD), etc might suggest that the converse proposition is true.

Seen from the history of digital games, the games were initially designed for competing with other player(s) due to lack of digital game related AI algorithms. However, these games do not belong to the category of the MODIGs because multiple players are connected through electronic circuitry.

The term 'online' denotes that communications among players occur through network. Therefore, the debut of Multi-User Dungeon (MUD) games is considered as the first appearance of MODIGs. MUD is usually a text-driven digital game in which multiple players can explore the virtual world to solve mysteries and achieve goals. The first MUD was programmed in 1979 by two students in Essex University, Roy Trubshaw and Rechard Bartle. DEC system-10 was used and it was connected to ARPANET through an experimental packet-switching system (EPSS).

Habitat was one of early graphic-based MUD games which achieved prominent success on the digital game market. It was developed by Randy Farmer and Chip Morningstar for Lucasfilim in 1985 and released in 1987 by Quantum Link for Commodore 64 computers through America Online. Since the advance of technologies related to 3D rendering, CPU process power, networking, etc, the virtual environment of digital games move toward VR-style environment. Most recent Massively Multiplayer Online Role-Playing Games (MMORPG) can handle more than hundreds of thousands of players simultaneously through server clustering technology and they provide persistent worlds where evolve without rest.

The persistency of the game world is one of main characteristics of MMOGs. It means that the status of players is kept somewhere, usually in server side, and it can

be retrieved without loss, damage, or alternation whenever the players log into the game world. It also denotes that other players will gain experience and grow up in the virtual world while someone is out of this world. One of the main goals of RPG games is the improvement of player's character, which is called avatar, in terms of its ability in the virtual world. Therefore, the persistency is one of the main reasons of game addiction on MMOGs. Also, this implies that one of the primary reasons for the preference of Client/Server (C/S) architecture to Peer to Peer (P2P) on the choice of network architecture of MMOGs. Detailed reviews on representative MODIGs such as Doom, StartCraft, WOW, Cart Rider, and BnB will follow in the next section and network architectures on MODIGs will be covered in Section 2.2.6.

2.1.3. Genre of Games

The MODIGs can be categorised into many groups according to their features. The general categories can be found in (Wolf, 2002):

- Role-player games (RPG)
- First person shooter (FPS)
- Adventure
- Sports
- Racing
- Action / Arcade
- Puzzle
- Simulation

- Real time strategy (RTS)

However, the recent trend of merging several genres into one game makes difficult to identify games clearly. Therefore, to show the distinguishable features, several representative examples are explained in detail in the following section.

Doom

The first example is 'Doom' which achieved amazing success on the market of MODICs. One of main reasons to explain its popularity comes from the variety of play strategies, which were not available at the time human users played with AI controlled computer. Doom was released in 1993 by ID Software as a shareware, so that users could try the software for free before they decide to purchase it. Shareware may limit some functionalities of the software until users purchase full licence. 15 million copies of this game were downloaded around the world and more than 250,000 people have registered for the full product (id Software, 2003).

Doom is a typical first person shooter (FPS) game which shows first-person perspective on the virtual game world. A player could virtually act as a marine soldier in a planet and tries to find exit and survive from the battles with unknown monsters. One of the tasks in the game is to find armours and weapons so that players could protect from the attack of the creatures and finally destroy enemies. Figure 2.2 shows the screen shot from episode-one of the game.

Figure 2.2: Doom (Wikipedia – Doom, 2006)

Doom supports not only human vs. computer mode but also various human vs. human modes such as co-operative and death-match modes up to two players via modem or serial link and maximum four people via IPX network or LAN. In co-operative mode, players co-operate and help each other fight the monsters. In the death-match mode, players fight each other with or without monsters in the virtual environments.

To maintain consistency of game states between players, this game uses a frequent transmission scheme which sends player state packets frequently and regularly (Singhal et al., 1999). This routine is performed even when there is no change in states, wasting network bandwidth and restricting the maximum number of players in network games. Doom does not utilise dead reckoning algorithms either. Thus, it flooded LANs with packets at a frame rate.

StarCraft

In 1998, Blizzard Entertainment released StarCraft and it was estimated that 9 million copies were sold until 2005 including BroodWar, the expension pack of StarCraft. This game is a typical real-time strategy (RTS) that the results of the victory or defeat on the game will be decided by strategies of each player. The strategies are mainly for

gathering resource, managing units and buildings, and controlling combats. Usually, resources in the RTS games are limited and constructing buildings and units require the limited resources. In the StarCraft game, there are two kinds of resources: mineral and Vespene gas, and the resources are gathered by worker units. There are three different types of tribes: Terran, Zerg, and Protoss, in which player could fight each other for survival. Each tribe has own weapon system, technologies, and different graphic design for buildings and units, all of which make the game more interesting because balancing of skills and power among the tribes is well configured and adjusted through well maintained patches.

Maximum eight players are supported by using various network means such as LAN, Dial Up, and IPX (Internetwork Packet Exchange) network. The main difference in network systems between Doom and StarCraft is the adaptation of StarCraft's own dedicated game server system, Battle.Net, which works as a game lobby system that provides game session related services such as creating, listing, and joining game sessions for players all around the world and supplies ladders for advanced players. Battle.Net only works as an intermediate system which means every player has to gather in the lobby to initiate a new game session, which finishes after players depart from the lobby. After the completion of the game session, the game result is sent to the lobby system. The snapshot of replay screen is displayed in Figure 2.3. Through the replay functionality, players can observe their plays later on and others also can learn from the replay.

Figure 2.3: StarCraft (Wikipedia – StarCraft, 2006)

World of Warcraft (WOW)

World of Warcraft (WOW) is a Massively Multiplayer Online Role-Playing Game (MMORPG) and it is the online version of Blizzard Entertainment's Warcraft series. The original genre of Warcraft series was RTS but Blizzard decided to create a MMORPG version of Warcraft and this game also recorded great hit on game market and more than six million players have subscribed for this game in the worldwide (Blizzard, 2006).

As it is mentioned in the previous section, an MMORPG is a place where you explore with other hundreds of thousands people simultaneously. The persistent world exists independent of your presence, and in that players' actions can permanently change the world. WOW is accessed through an Internet connection, giving players the ability to immerse themselves into the mystical world of Warcraft with other players from all around the world. There are various races in the game that participants can choose such as Human, Night Elf, Orc, Undead, and so on. Players can choose allowed classes such as Druid, Hunter, Mage, Paladin, Priest, and so on from their race and dozens of professions are provided in the game. Figure 2.4 shows the screenshot of the game.

Figure 2.4: World of Warcraft (Wikipedia – World of Warcraft, 2006)

Different from aforementioned games, such as Doom and StarCraft, WOW uses dedicated game servers, known as 'realms' for the persistent world. Up to Jun 2006, 168 realms are available worldwide and they could be divided into four types, they are: PvE (player versus environment, known as normal), PvP (player versus player), RP (role-playing), and RP-PvP (role-playing with player versus player). Interaction between players is only allowed with players who reside in the same realm and it is permitted to transfer their characters between realms but it is not free of charge at the moment.

Crazy Arcade BnB

In 1983, Nintendo published a Nintendo Entertainment System (NES) based game named 'Bomberman' which adapted the game concept from one of Hudson Soft's games, Eric and the Floaters. Only one player was supported in the Hudson Soft's original game and later on its second version, Bomberman II supported multiplayer functionality in 1992. Figure 2.5 shows the screenshot of the bomberman game.

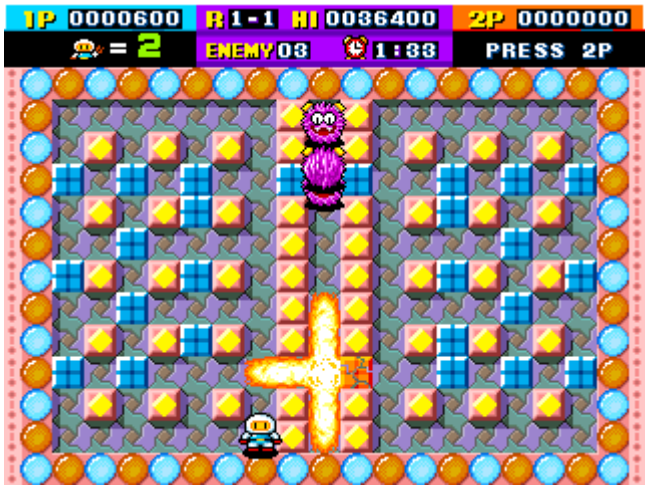


Figure 2.5: Bomberman (Wikipedia – Bomberman 2006)

One of Korean online game companies, Nexon, developed and serviced networking version of bomberman in 2003. The game name implies that the genre of this game is a kind of arcade/action game. Originally, the term 'arcade games' was used to depict games on coin-operated entertainment machines which installed in video arcades, shopping malls, etc. One of the main characteristics of arcade games is the duration of one game session. This should be relatively short compared to other genre of games as to make profit for the owners of game machines. This feature has constrained the genre of arcade game to fast-paced action games. BnB is a typical arcade game and it usually takes about 10 to 30 minutes to finish one game session. It is very short time duration compared to the time duration for achieving one of main game goals in MMORPG that is reaching maximum level of his/her character. Generally, it takes more than months or even years to reach the maximum level of the character in MMORPG games. Certain RTS game takes a few hours for reaching the end of game session. BnB also implemented a ranking system that shows overall, and ladder rankings to provide another game goal that makes the game addictive, which is achieving high score in games.

The main aim of BnB is destroying all other enemy players using water bubbles. Each player can team up with other players against enemy groups or play individually. When a player is captured in a water bubble then same team members can rescue the player by blasting the water bubble or enemy players can blast the water bubble to destroy the player. There is also time-out system. If the fixed time elapses without rescuing the same team members who are captured in the water bubble then the player will be destroyed. The control and operation method of this game is relatively simple and intuitive compared to other genre of games. BnB supports eight types of characters and various items.

Maximum eight players can play the game together at the same time through the Internet connection. Nexon provides its web site as a connection point. They require authentification information for logging into the game system and it can be obtained through membership service from the web site. They also provide a hybrid P2P system consisting of a lobby system and a P2P system, which supports C/S network architecture for game creation and joining, and P2P network architecture for game playing.

CrazyRacing KartRider

Nintendo released the first version of Mario Kart series, Super Mario Kart, on Super Nintendo Entertainment System (SNES) in 1992. This game series is one of most popular digital racing games. It supported maximum two players and provided various racing tracks and unique eight characters, Mario, Luigi, Yoshi, Princess Toadstool, Koopa Troopa, Toad, Donkey Kong Junior, and finally notorious Bowser, which were used in their famous games, Mario series. Mario Kart DS, the latter version of this game on Nintendo DS (NDS or simply DS), can support maximum eight players

through wireless 802.11b (WiFi) network. Figure 2.6 shows the screenshots of Mario Kart DS.

Figure 2.6: Mario Kart DS (Wikipedia – Mario Kart, 2006)

The publisher of BnB, Nexson, started online game service for cart racing game named, KartRider, on the Internet in 2004. According to the report from the company in 2005, there were about 220 thousands concurrent players and the number of members reached over 11 millions in South Korea. Revenue from selling items in this game is around 60 million dollars per month and this game is going to be serviced in Japan, China and other countries (Choi, 2005).

Same as the aims of other typical racing games, the aim of this game is to finish the goal line before other players do. Players can choose one of characters from currently available 15 characters, most of which have slightly modified designs from Nexon's popular game, BnB. Figure 2.7 shows the screenshot of the game.

Figure 2.7: A screenshot of KartRider (KartRider, 2006)

The aforementioned two games, BnB and KartRider, are typical casual games which are developed in South Korea. The common features of them are:

- The maximum number of supported players in one game session is relatively small compared to MMOGs;
- The game duration of one game session is short;
- They provide easy control and operation; and
- They are built on P2P-based network architecture.

2.1.4. Challenges

Network bandwidth and latency are important factors in achieving playability in MODIGs. On one hand, because of the recent widespread of broadband internet access, the network bandwidth problem is not a big issue any longer. On the other hand, network latency problem is hard to solve due to the physical characteristics related to data transmission. The latency in typical Ethernet LAN is generally less than 0.3ms and varied in WAN environments (Cheshire, 1996a). The main cause of network latency is the distance between players, but network situation such as congestion can be a major contributing factor. According to Cheshire's experiments (1996a), it took 84.5 milli-seconds (ms) for a round-trip from the east coast to the west coast of the United States. Usually, round-trip time (RTT) for international transmission is longer. According to the preliminary experiments performed for this research, it took about 300 ms for a round-trip from Gold Coast, Australia to Seoul, Korea (Moon et al., 2005). We will examine methods that improve network latency by sacrificing network bandwidth in chapter 5 and 6.

Another factor that will affect playability is the game architecture. Network architectures of MODIGs can be divided into two categories: Client/Server (C/S) and Peer-to-Peer (P2P). The centralised game architectures are the C/S based and widely used for most MMOGs due to the simplicity of consistency maintenance, better security, improved authentication, and easy billing system (Moon et al., 2005). However, this architecture introduces additional network latency compared to distributed game architectures and this can cause a network bottle neck due to lack of network resources. Distributed game architectures can eliminate redundant packet transmissions between server and client as to reduce network latency. Because each

node manages its own object states, it suffers from the hardship of maintaining consistency of game states between players due to network delay (Jiang, 2005).

The current networking issues, concurrency control algorithms, and resource management techniques for the problems in MODIGs will be examined in the following sections.

2.2. Networking Issues in MODIGs

For exchanging data and information between participated players in a session of games, network is utilised as a medium in MODIGs. In building robust networking modules of MODIGs, latency, bandwidth, reliability, and responsiveness are the most significant aspects to be considered.

2.2.1. Network Latency

In general, most of MODIGs exchange messages through packet-based network. The end-to-end delay between hosts is referred to as latency and it has significant impacts on users' playability. Jehaes et al. (2003) defined the term latency in MODIGs as "*the sum of the computational delay introduced by the game consoles and game servers themselves, and the delay incurred in the transport of the information flow associated with these games over the network.*"

Many researchers have pointed out that network latency is a major cause of problems in actualisation of MODIGs (Armitage, 2003; Cheshire, 1996b; Harvey, 1997; Jiang et al., 2005; Pantel & Wolf, 2002). Cheshire (1996a) also mentioned that network bandwidth problems can be addressed by increasing a number of connection lines with own networking software that is a similar concept as one of using multiple ISDN

channels in parallel. He enumerated various causes of latency such as packets propagation speed, data processing speed in nodes (e.g. server, client, router, hub, and switch), and waiting time in queue (or buffer). Among them, he indicated that problems caused by latency are hardly addressable mainly due to limited packet propagation speed, which is currently approximately 200,000 km per second in optical fibre. With this speed, it takes about 200 ms to transfer a message to opposite side of the earth.

Cheshire (1996b) suggested 100 ms as a target Round-Trip Time (RTT) for acceptable interactive applications, which is latency of 50 ms for packet propagation. There are many qualitative and quantitative researches of correlation between latency and the playability of genre of MODIGs, such as first person shooter (FPS) games (Beigbeder et al., 2004; Quax et al., 2004), massively multiplayer online role-playing games (MMORPGs) (Fritsch et al., 2005), racing games (Yasui et al., 2005), real-time strategy (RTS) games (Sheldon et al., 2003), sports games (Nichols & Claypool, 2004). The experimental results show that the acceptable latency varies from 60 ms for fast action games to even several seconds for RTS games due to their characteristics related to interactivity. Sheldon et al. (2003) explained the insignificance of latency on RTS games as its emphasis on strategy rather than interactivity.

Beigbeder et al. (2004) reported that latency higher than 100 ms causes high failure rate of shooting actions in one of most famous FPS games, unreal tournament 2003. Quax et al. (2004) experimented on the same game and concluded that users have started to notice latency from 60 ms. Pantel and Wolf (2002) implemented their own car racing game and pointed out that more than 100 ms of latency should be avoided

to provide acceptable playability for racing games. They also provided qualitative research results as a form of survey of subjective impression. According to the results, latency up to 100 ms is acceptable and hardly detectable if there is no need to provide high interactivity. However, 200 ms of latency was clearly observable and overall behaviour was not realistic.

The delay in packet-based network can be categorised into propagation delay, packet processing delay, serialization delay, and queuing delay (Jehaes et al., 2003). The propagation delay is the elapsed time to propagate a message between each link of the networks. Packet processing delay is the sum of necessary calculation time for packet error checking, preparation of packet relay, etc. The serialisation delay is the time taken to put all data from header to trail on the link when packets are generated in sending hosts. It is proportional to the size of packets and reciprocal to the bandwidth (or data rate) of the link. Finally, the queuing delay is the waiting time in queue (or buffer) to be processed further.

Every network device causes minimum latency, which is called as the latency of device (Cheshire, 1996b). Latency from typical Ethernet connections is about 0.3 ms and communications with modem yield 100 ms of latency. Ng (1997) mentioned that additional 2 ms of latency per one byte occurs in 14.4 Kbps modem and approximately 56 ms of latency is not avoidable disregarding the size of messages in modem communication. This 56 ms latency is caused by modem compression algorithm, queuing delay, line equalisation. He also enumerated various sources of latency as shown in Table 2.1.

Table 2.1: Communication Latency

Source of latencies	Possible, corresponding solutions
TCP layer buffering	Carefully tune TCP settings, which are set by the setsockopt() call.
Thread priorities on both client and server	Use threads carefully and manage CPU use.
Speed of light (propagation delay)	Sorry, We haven't beaten Einstein yet.
Store and froward routers	Use a well-managed network. Avoid routers.
Over-used routers causing packets to be dropped and retransmitted	Use a well-managed network.
Number of routers between source and destination	Use a strategically located, distributed set of servers.

Network latency is closely related to response time, especially when pessimistic concurrency control mechanism is adopted for maintaining consistency of entities in MODIGs. The response time is the delay between user's actions as an input and response as an output. The quickness of response is referred to as responsiveness. Generally, the pessimistic approaches utilise lock-based algorithms such as lockstep to assure the safeness of user's actions to commit. Therefore, latency is inversely proportional to responsiveness in MODIGs with the pessimistic approaches.

To improve responsiveness by masking network latency, various optimistic concurrency control methods are proposed. However, the optimistic approaches may harm consistency of entities' states due to unsynchronised packet propagation delay. Inconsistency of game states can heavily affect the playability of MODIGs. Cheshire (1996a) proposed two approaches to improve the quality of interaction in MODIGs. First, remove all unnecessary latency in computer hardware and software. Second, develop methods to mimic (or fake) interactivity to hide network latency if true interactivity is not possible. Dead Reckoning (DR) algorithm and bucket synchronisation (BS) algorithm belong to the latency hiding (or lag compensating) methods (Bernier, 2001). Aforementioned pessimistic and optimistic approaches are explained in detail in the following sub-sections.

2.2.2. Network Bandwidth

Bandwidth, also referred to as data (or link) rate, is the transmission capacity of a network line and it denotes the amount of data received (inbound) and/or transmitted (outbound) per unit time (Smed et al., 2001).

The bandwidth varies depend on the type of connection lines and hardware devices. Local area network (LAN) can support much larger bandwidth than wide area network (WAN). Generally, the range of bandwidth in LAN is from 10 Mbps (mega bits per second) to 10 Gbps (giga bits per second) while the bandwidth of WAN can be (Peden & Young, 2001): dial-up modems (up to 56 Kbps), up to 144 Kbps of Integrated Services Digital Network (ISDN), Digital Subscriber Line (DSL – see Table 2.2), cable modems (up to 30 Mbps), T1 (up to 1.5 Mbps), and T3 (44.7 Mbps).

With recently developed Wavelength-Division Multiplexing (WDM) technique, service providers are deploying Synchronous Optical Network (SONET) of Optical Carrier (OC) level 192 with bandwidth of 10 Gbps (Chatterjee & Yang, 2005) and it is currently the largest bandwidth available on the Internet (Kornaros et al. 2003). Bit Error Rates (BER) of typical copper wire and optical fibre links are 10^{-4} and 10^{-9}, respectively (Bangun & Beadle, 1997). The BER is denoted by the ratio of the number of incorrect bits to the total transferred bits per unit time.

Bandwidth requirement denotes the amount of data required for nodes to handle, not to delay processing and/or cause harm on overall performance. When bandwidth requirement is larger than bandwidth of the link, packets can be discarded from queue in router due to queue overflow or processing can be delayed by the dropped packets.

Table 2.2: Copper Access Bandwidth

Transmission Systems	Bandwidth
ADSL – Asymmetric Digital Subscriber Line	2 wires for up to 6 Mbps downstream, 800 Kbps upstream
HDSL – High Bit Rate Digital Subscriber Line	4 wires for 1.5 Mbps channelised symmetric service
HDSL2 – High Bit Rate Digital Subscriber Line v.2	2 wires and a single pair version of HDSL
VDSL – Very high bit rate Digital Subscriber Line	2 wires for up to 53 Mbps asymmetric, or 13 Mbps symmetric
ISDL – Integrated Services Digital Network	2 wires up to 144 Kbps symmetric service

The major factors affecting the bandwidth requirement of nodes are as the followings: the number of users, communication architecture (e.g. client/server, peer-to-peer, peer-to-peer with lobby server, and client/server with distributed servers), and transmission techniques (e.g. unicast, broadcast, multicast). A Client/Server (C/S) architecture employs a central server (or multiple servers known as server clustering technique) to control and manage client hosts. A Peer-to-Peer (P2P) architecture contains hosts which have equal authorities and roles in the networking communication session. In unicasting, a message from a sender is transmitted to a targeted recipient only. A broadcasting transmission technique can disseminate sender's message to everyone in the same network. In multicasting, multiple target recipients can be selected by specifying their multicast group IP (Internet Protocol) address (Smed et al., 2001). More detailed explanation on aforementioned factors will be presented in following sections.

Bandwidth is one of main factors that can limit the number of concurrent users in MODIGs. Scalability, the capability of adaptation to the changes of resources, is tightly coupled with bandwidth due to this reason. Therefore, reducing bandwidth requirement is an important task for the actualisation of MODIGs. As mentioned

before, communication architectures can affect not only network latency but also the bandwidth requirements of game participants (Pellegrino & Dovrolis, 2003).

Bandwidth requirement reduction techniques can be divided into communication architecture, interest management, packet compression, and packet aggregation based approaches. The relationship between communication architectures and bandwidth requirement is explained in Section 2.1.5. Interest management techniques (Barrus et al., 1996; Funkhouser, 1996; Greenhalgh & Benford, 1997; Huggahalli et al., 2005; Macedonia et al., 1995; Singhal & Chriton, 1996) reduce bandwidth requirements of nodes by filtering packets. Packet compression methods (Bormann, 2001; Chattopadhyay et al, 2005; Taylor et al., 2005; Tye & Fairhurst, 2003) reduce overall packet size by lowering the redundancy of data. Packet aggregation (Kanamaru et al., 2000; Makki, 2006; Razafindralambo et al., 2006; Singhal, 1996; Watagodakumbura, 2003) approaches merge multiple packets into one to decrease the overhead of packet headers. The latter three categories are presented in Section 2.4.

2.2.3. Network Reliability

Certain types of network applications require absolute accuracy of data transmission. Network reliability is more important notion than that of the timeliness of data transmission to these types of network applications. According to Medhi's (1999) definition, "*network reliability refers to the reliability of the overall network to provide communication in the event of failure of a component or components in the network*".

Various causes such as hardware and software malfunctions, accidental cable cut, natural disasters, and malicious attack can cause data drop and curruption (Medhi,

1999). These data drop and corruption rates are used for the measurement of network reliability and these two terms are denoted as data loss. Data drop happens, for example, when routers receive more packets than they can handle. Also, it may happen if receivers' computational power or bandwidth for packets being transmitted from senders is not sufficient enough to handle all packets (Singhal & Zyda, 1999).

Single-bit error or burst error can occur during data transmission and these can alter/corrupt the original messages. In general, these errors are caused by noises which are added into original messages. As the name implies, single-bit error happens when only one bit of data is changed. In parallel transmission, the single-bit error can be happen more often than in the case of serial data transmission. The single-bit error is rare in serial transmission, especially for high bandwidth network because the duration of noise is much longer than that of 1 bit. When two or more bits are altered by communication error, it is called as burst error (Forouzan, 2006).

When data drop or corruption occurs, packet retransmission may be required to improve network reliability. Error detection and correction mechanisms are used for examining the integrity of packets and rectifying the errors. These two groups are called as error control and methods which are broadly used for error control are as follow: error detection using Cyclic Redundancy Check (CRC), positive acknowledgement, retransmission after timeout, negative acknowledgement and retransmission (Stallings, 1997). CRC is a redundancy checking technique which is based on binary division. In CRC, binary data which will be transmitted is divided by predetermined binary number and its remainder, CRC remainder, is added at the end of the original data. The destination node checks if the data is intact by performing

binary division. When the original message is found to be altered, several steps will be followed for retransmission processing if necessary (Forouzan, 2006).

The acknowledgement messages sent by receivers after senders' packets arrive at destination hosts successfully are known as positive acknowledgment. The positive acknowledgement packets may not be received in determined period, and then senders assume that the packets are dropped and retransmit the packets. This type of methods is called as retransmission after timeout. As opposed to positive acknowledgment methods, negative acknowledgement and retransmission approaches disseminate negative acknowledgement messages to senders if transmitted data contains faulty. When senders receive the negative acknowledgement, they retransmit the requested messages from the recipients. These mechanisms are called as Automatic Repeat reQuest (ARQ) and reliable network transmission is enabled through these methods (Singhal & Zyda, 1999).

However, ARQ may cause significant processing overhead for those like error detection and correction, retransmission, flow control, and transmission ordering. Transmission reliability may be more meaningful than timeliness of packet delivery in certain applications such as file transfer, email, and online transaction. Video and audio streaming softwares are opposite examples that require high timeliness of message dissemination. Also, most real-time applications such as CVE, NVE, and MODIGs prefer timeliness to reliability. When reliability is preferred, Transmission Control Protocol (TCP) is broadly used for propagations of messages in the Internet. On the opposite case, User Diagram Protocol (UDP) or Real-time Transport Protocol (RTP) are adopted to reduce performance overhead and increase timeliness of

transmission. Major Internet protocols for MODIGs are explained in the following section.

2.2.4. Internet Protocols for MODIGs

A communication protocol means rules for establishing data connection and information exchange between senders and recipients (Kim, 2000).

The basic elements of protocol are:

- Syntax: data structure, coding, signal level
- Semantic: Coordination and control code for error correction
- Time: Speed control, ordering

Transmission Control Protocol (TCP), User Diagram Protocol (UDP), and Real-time Transport Protocol (RTP) are broadly used protocols for MODIGs on the Internet. TCP is a reliable connection-oriented protocol. This protocol establishes a connection between two end nodes and transmits packets through the connection. UDP is an unreliable connectionless protocol. RTP is also, unreliable protocol which is broadly utilised for real-time audio and video on the Internet. Including aforementioned protocols, transmission techniques such as unicasting, broadcasting and multicasting are discussed in the following sections.

2.2.4.1. TCP (Transmission Control Protocol)

TCP is a reliable, point-to-point connection-oriented, stream-oriented protocol which utilises flow control, error control, and congestion control for transmitting data through network (Forouzan, 2006).

Flow control is a processing of controlling data transmission not to overwhelm recipient's buffer. TCP uses a sliding window protocol for the flow control that is similar to a dynamic buffer. The size of window can be dynamically changed according to the network and processing situations of recipient.

Error control is used for error detection and error correction of corrupted segments, lost segments, out-of-segments, and duplicated segments. Checksum, acknowledgement, and retransmission after time-out schemes are used for error detection and correction.

Congestion control is an important mechanism that can affect network latency. When sender's transmission data rate is larger than the capacity of network devices such as routers, packets are being waited in the queue of the devices that results the increment of queuing delay. For congestion control, TCP utilises congestion window which determines sender's transmission data rate. For the determination of congestion window size, three congestion control phases - slow start, congestion avoidance, and congestion detection, are used. As the name implies, slow start algorithm sets the size of congestion window to one maximum segment size (MSS), which is minimum size that can be sent at once. When the transmission is successful, the congestion window size increases exponentially until it reaches the threshold of slow start algorithm. After that, the size of congestion increases additively until congestion is detected. Depends on the congestion situation, the size of congestion window becomes one MSS (slow start) or half of the previous size (multiplicative decrease).

The advantages of TCP are such as guarantee of packet delivery, preservation of packet ordering, transmission flow control, error detection and correction, and

supports from most firewalls. However, TCP mostly supports point-to-point connection only. Also, bandwidth requirements of TCP can be higher compared to that of UDP. Most of all, TCP may cause longer packet propagation delay and higher processing cost due to the overhead of packet ordering and congestion control mechanisms (Singhal & Zyda, 1999). To support aforementioned control mechanisms, TCP header includes additional fields compared to that of UDP. Figure 2.8 shows the structure of TCP header and explanation on them follows (Forouzan, 2006).

One of most popular Real-Time Strategy (RTS) games, Warcraft III, utilises TCP as its packet transmission protocol. Generally, RTS games do not require fast response time due to its emphasis on strategy rather than interactivity (Sheldon et al., 2003). Therefore, TCP may be suitable for these types of MODIGs but most of MODIGs which require real-time interaction rather than strict packet ordering and reliability would not be suitable for TCP.

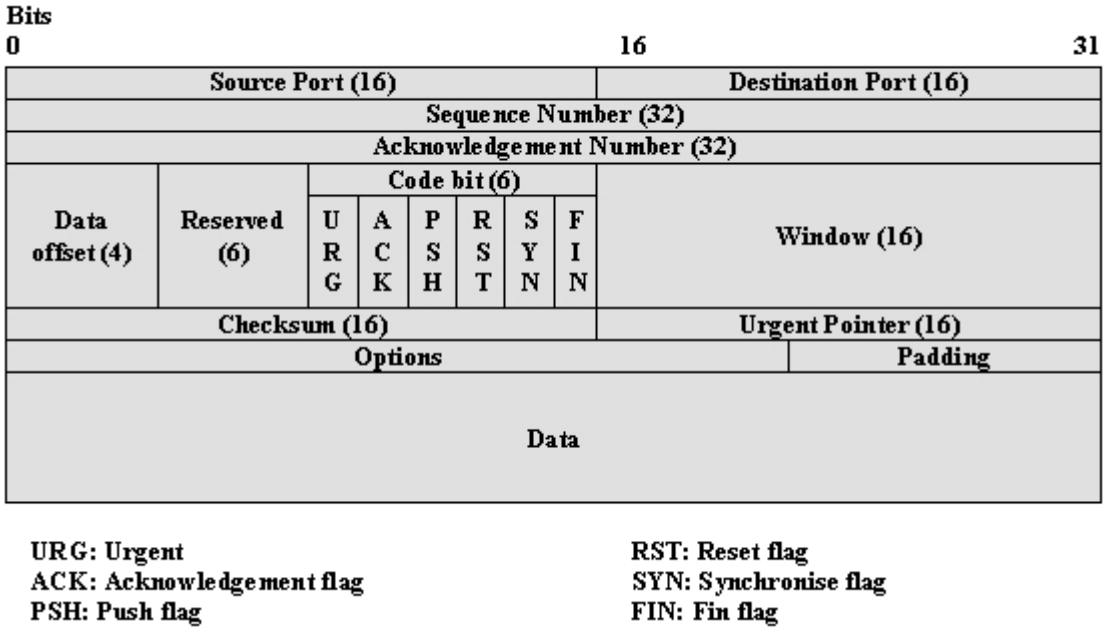


Figure 2.8: TCP Header and Data

- **Source port (16 bits):** Source service access point.

- **Destination port (16 bits):** Destination service access point.

- **Sequence number (32 bits):** Sequence number of the first data octet in this segment except when SYN is present. If SYN is present, it is the initial sequence number (ISN), and the first data octet is ISN + 1.

- **Acknowledgment number (32 bits):** A piggybacked acknowledgment. Contains the sequence number of the next octet that the TCP entity expects to receive.

- **Data offset (4 bits):** Number of 32-bit words in the header.

- **Reserved (6 bits):** Reserved for future use.

- **Flags (6 bits):**

  URG: Urgent pointer field significant.

  ACK: Acknowledgement field significant.

  PSH: Push function.

  RST: Reset the connection.

  SYN: Synchronise the sequence numbers.

  FIN: No more data from sender

- **Window (16 bits):** Flow control credit allocation, in octets. It contains the number of data octets beginning with the one indicated in the acknowledgement field that the sender is willing to accept.

- **Checksum (16 bits):** The one's complement of the sum modulo 2^{16}-1 of all the 16-bit works in the segment, plus a pseudo-header. The situation is described below.

- **Urgent Pointer (16 bits):** Points to the octet following the urgent data; this allows the receiver to know how much urgent data are coming.

- **Options (Variable):** At present, only one option is defined, it specifies the maximum segment size that will be accepted.

2.2.4.2. UDP (User Diagram Protocol)

UDP is a simple, unreliable, connectionless data transmission protocol which supports limited functionality compared to TCP (Forouzan, 2006). UDP is described in Internet as Engineering Task Force (IETF) Request for Comments (RFC) 768. UDP does not guarantee packet transmission nor examine packet replication (Postel, 1980). However, many MODIGs adopt UDP as a data transmission algorithm because of its low overhead of sending small size message frequently.

The advantages of UDP are low overhead of packet processing, and immediate transmission because of the absence of flow control, error control, and congestion control mechanisms. The simplicity of UDP is the main reason of the popularity of this protocol to most MODIGs. However, UDP also support only point-to-point connection so packet error detection may become impossible when checksum field is

not used (Singhal & Zyda, 1999). Furthermore, most of firewalls are set to block UDP packets due to potential dangers of malicious attacks on network and its popularity on MODIGs. Especially, majority of educational organisations ban the playing of MODIGS in computer labs. Simply, network administrators in organisations could achieve the aims by blocking UDP packets in their network.

As shown in Figure 2.9, UDP header contains minimum fields for data transmission. Source and destination ports are used for process-to-process communication. A running application is called as a process (Forouzon, 2006). Each network application on a host requests a port number to its Operating System (OS) and the port number is used to distinguish processes in the host. The length field recodes sum of header and data length. The checksum field is used for error detection but it is optional in UDP that is different from TCP. Even checksum is calculated and if error is detected, the packet will be discarded but no other procedure will follow (Stallings 1997). Therefore, if reliability of data transmission is required in MODIGs without significant overhead of packet processing and transmission delay, designers of MODIGs need to implement their own network codes based on UDP (Singhal & Zyda, 1999).

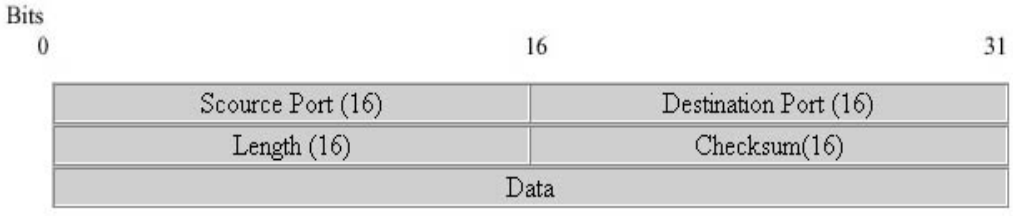


Figure 2.9: UDP Header and Data

2.2.4.3. RTP (Real-time Transport Protocol)

RTP protocol is used to transfer real-time data such as interactive audio and video (Schulzrinne et al., 1996). Generally, real-time applications such as MODIGs, audio/video conferencing, and streaming audio/video run RTP on UDP to disseminate data in timely manner. However, RTP can also run on other transport protocols to provide multicast functionalities. Unicasting is a data transmission technique that sends packets to one destination at a time. In broadcasting, packets are disseminated to all hosts in the same network. Multicasting allows transmitting packets to a group of hosts. These transmission techniques will be discussed in the later sections.

Compared to UDP, RTP includes additional information in its header such as version, sequence number, timestamp, payload type, synchronization source (SSRC) identifier, and contributing source (CSRC) identifier. Figure 2.10 shows the header structure of RTP and explanation of each field follows (Schulzrinne et al., 1996).

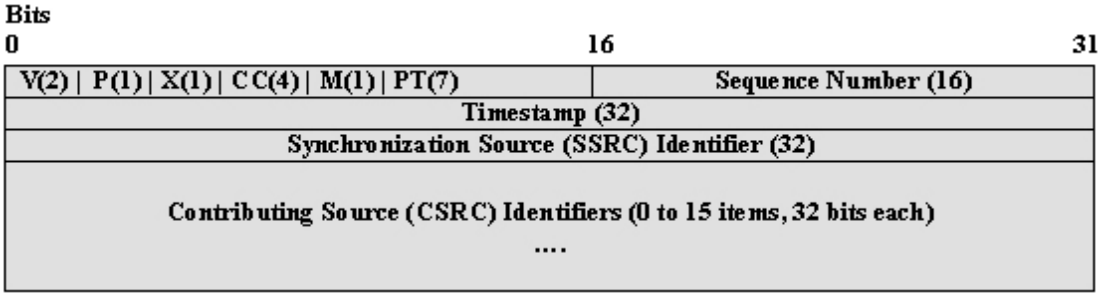


Figure 2.10: RTP Header Structure

- **Version (V):** The version of RTP. The current version is 2.

- **Padding (P):** If this value is 1, there are additional padding octets at the end. This information may be used by encryption algorithms.

- **Extension (X):** When it is set, a variable-length header extension must be added to the header, otherwise fixed-size header is used.

- **CSRC count (CC):** The number of CSRC identifiers.

- **Marker (M):** To be used for notifying significant events such as frame boundaries to be marked in the packet stream.

- **Payload type (PT):** To identify the format of the RTP payload such as the type of video codecs.

- **Sequence number:** The identification number of RTP packets. It increases by one and initial value should be randomly generated number. This information can be used for packet ordering and packet drop detection.

- **Timestamp:** The generation time of first octet in RTP data. Its initial value should be random as the sequence number. This value can be used for delay jitter detection.

- **SSRC:** A unique identification code which is randomly generated for synchronization source.

- **CSRC list:** Maximum 15 items can be inserted. These values identify the contributing sources for the payload contained in the packet.

In conjunction with RTP, RTP control protocol (RTCP) is used to monitor data delivery. Quality of Service (QoS) feedback is delivered to the source node for monitoring and congestion control. The QoS feedback information includes the number of lost packets, round-trop time, and jitter (Liu, 2000).

MiMaze, a packman-like 3D Internet game, uses RTP over UDP/IP multicast for data transmission (Diot & Gautier, 1999; Gautier & Diot, 1998; Gautier et al., 1999). The network architecture of MiMaze will be explained in Section 2.2.6 in detail. Mauve et al. (1999) proposed RTP/I which is a real-time application-level protocol based on RTP, specifically used for real-time interactive applications such as CVE, NVE, and MODIGs. RTP/I provides a standardised frame structure and common functionalities which are required to CVE, NVE, and MODIG applications such as the ability to record sessions, support of late joining to sessions, and security services. Vogel and Mauve (2001) utilised RTP/I for their multimedia lecture board application to exchange real-time messages.

Perkins and Crowcroft (2000) criticised that RTP may not be suitable for shard workspace applications due to the complication of interactive applications such as MODIGs. The object state in MODIGs can be based on not only time but also user interaction. Therefore, resolving confliction between diverse states between participated hosts in game sessions is important but RTP itself does not support inconsistency resolution. They also mentioned that RTP/I doubled the header size of RTP and state queries performed by a new RTCP packet type may degrade the performance of RTP/I.

2.2.5. Transmission Techniques

In this section, three transmission techniques, namely unicasting, broadcasting, and multicasting are explained. Unicasting is a transmission technique that sends data to one host at a time. In broadcasting, all nodes in the same network receive packets from hosts which broadcast their data. Multicasting can group nodes and data can be

disseminated to particular groups which are chosen by origin hosts as destination groups.

2.2.5.1. Unicasting

Unicasting is a transmission technique that sends packets from one host to another host at a time (Legout et al., 2001). It is based on point to point communication which is a communication between processes which are running applications in a system. Each process waits for messages through a port (or ports) and unique port numbers are assigned to the process for identification purpose (Forouzan, 2006). TCP and UDP are typical transmission protocols for unicasting.

The most advantageous characteristics of unicasting are its simplicity of implementation and broad acceptance of unicasting protocols in network devices such as routers, hubs, and switches. The most prominent disadvantage of unicasting is that unicasting requires more bandwidth than multicasting and broadcasting. Each message must be copied and transmitted to multiple nodes in same game sessions even the message's contents are same.

Legout et al. (2001) suggested that two variables can be used for the determination of the choice of unicasting or multicasting. These are the number of receivers of the data and Round Trip Time (RTT). When the number of receivers of the data is considerably less than the total number of participants, the message can be sent by unicasting. Also, if there is any link that has remarkably long network latency, unicasting can be used to disseminate messages between the two nodes for reducing the overhead of multicasting. Figure 2.11 shows the model of unicasting communication.

The bandwidth requirement of unicasting in Client/Server (C/S) and Peer-to-Peer (P2P) architectures in MODIGs is explained in Section 2.2.6.

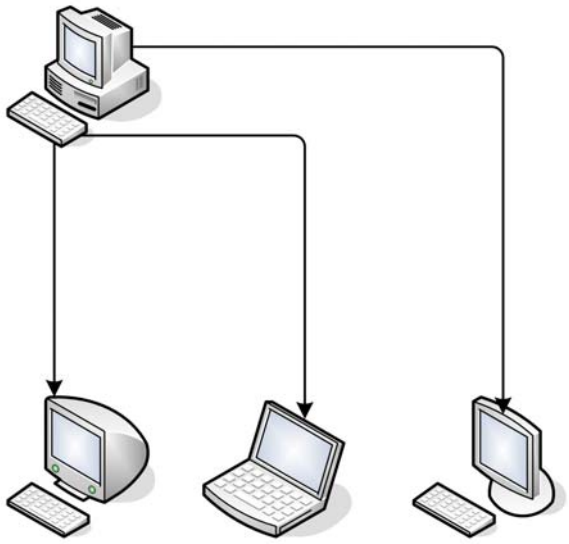


Figure 2.11: Unicasting Communication

2.2.5.2. Broadcasting

In broadcasting, packets sent by hosts are transmitted to every node in the same network. So, the broadcasting communication can significantly reduce bandwidth requirement of senders. However, the scope of broadcasting communication is limited to the local network. Also, several computer networks such as X.25 and frame relay do not accept broadcast packets (Broadcasting, 2006). Therefore, this approach is hardly used as a communication method in MODIGs except LAN-based ones.

IP broadcasting protocol is based on UDP and it can transmit data to all the nodes in the same network by sending a single message at a time. The IP broadcasting protocol is neither reliable nor duplication-free same as UDP (Mogul, 1984). It is suitable for network communications in a small-sized LAN which can not support a large capacity of bandwidth. For example, when TCP or UDP is utilised as a communication

protocol, *n-1* times of packet transmission is required where n is the number of participants in game sessions. However, only one transmission is necessary to disseminate the packet to every player in game sessions when IP broadcasting protocol is adopted. The model of broadcasting communication is shown in Figure 2.12.

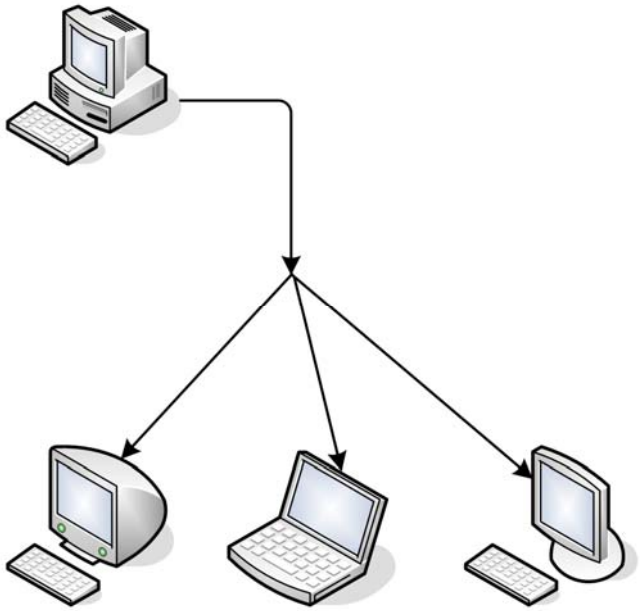


Figure 2.12: Broadcast Communication

The IP broadcasting protocol utilises UDP and each packet is encapsulated into IP and then Ethernet or Token Ring type message. Then, recipients need to retrieve original UDP packets from received format reversely. This procedure requires additional processing power and time to verify whether the packet is destined for the nodes (Singhal & Zyda, 1999). As aforementioned, this protocol is designed for working in LAN environment and it is not suitable for MODIGs which target for being played in the Internet.

2.2.5.3. Multicasting

When unicast is adopted as a transmission technique, the bandwidth requirement of a server can be O(n) or O(n^2) depending on the availability of packet aggregation (Sheldon et al., 2003). This can be reduced to O(1) or O(n) when multicasting is utilised (Ratnassamy et al., 2006). In network level multicasting, a single packet from each node is propagated to all other nodes through multicast-enabled routers. One of the most well-known multicasting protocols is IP multicasting which proposed by Deering (1989).

IP multicasting protocol transmits datagram type messages to multiple nodes which combined with same IP address. The IP addresses for multicasting are class D IP addresses which are from 224.0.0.0 to 239.255.255.255. Among them 224.*.*.*, 225.*.*.*, and 232.*.*.* are reserved for special usage by Internet Assigned Numbers Authority (IANA) (Goyeneche, 1999).

The disadvantages of the IP multicasting protocol are high complication, low scalability, weak security, and unreliability. In IP multicasting, routers need to store the state of each multicast group. This increases the complication of protocol implementation and harms scalability. Its weakness of security is caused by the fact that unauthorised nodes can transmit packets to any multicast groups. This enables malicious attacks on the network and makes network management more complicated. Furthermore, each multicast group is required to be assigned a globally unique IP address and this can severely harm the scalability of the protocol. Finally, the IP multicasting protocol is based on the UDP protocol which is unreliable. Therefore, it is hard to provide high level of services such as reliability, congestion control, flow control, and security (Chu et al., 2000).

2.2.6. Communication Models

Generally, three main procedures engage to actualise MODIGs. These are procedures for accepting input user commands, calculating results depends on the user input, and projecting the results on users' output devices such as monitors, speakers, Playstation 2's shock pads, and so on. The communication models such as Client/Server(C/S) and Peer to Peer (P2P) are categorised by the place where these procedures occur (Funkhouser, 1995). Figure 2.13 depicts the communication models of C/S and P2P.

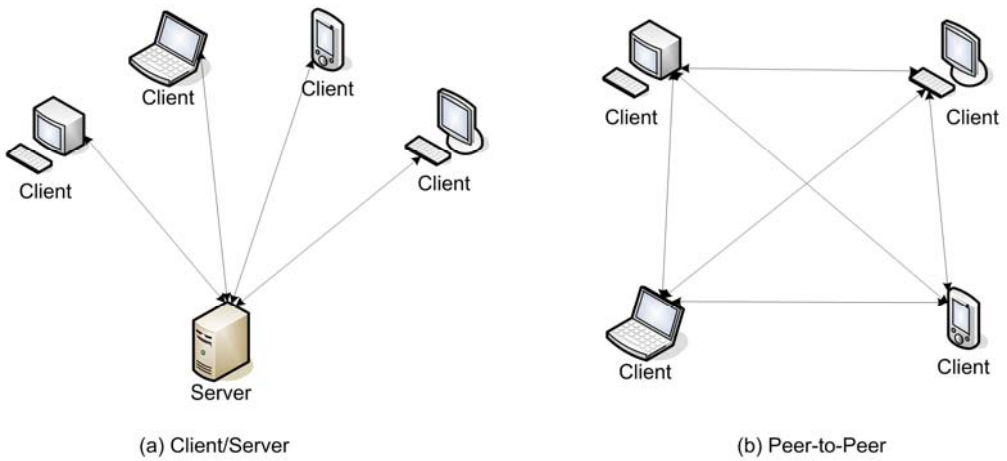


Figure 2.13: Communication Models

In C/S systems, user inputs occur on client sides and these inputs are transferred to the server side in the form of packets. A main game engine in the server processes inputs from clients to produce globally synchronised results among clients. These results are sent back to clients again and the procedure loop continues until the game session ends. Clients may hold their input procedure until the response from the server arrives and this can possibly prolong the network latency between the server and clients.

As shown in the figure above, P2P systems do not adopt a server for exchanging information in regards to their status. Instead, each peer possesses an identical game

engine. Using the game engine, each peer processes its input and propagates the results or the input itself to other peers. Therefore, the P2P systems may reduce network latency by removing the waiting time for central server's response. However, maintaining consistency algorithms are required and the bandwidth requirement of each peer is higher in the P2P systems than that of C/S systems (Jiang et al., 2005).

Following sections explain each system in detail. Also, hybrid systems which address the weakness of C/S and P2P systems are introduced.

A Centralised Model

In C/S systems, a single node possesses a game main engine and performs server roles such as authorises the actions of clients (Smed et al., 2001). The advantages of C/S systems compared with P2P systems are stronger security, easier administrative control, better flexibility, and consistency. On security, all clients inputs are processed in the server, there is less chance of cheating to clients. The server can easily maintain client applications when needed. Also, the server itself can be managed for upgrading and expanding. Maintaining consistency of the status of clients is an easy task compared to P2P systems because only the game engine in the server side generates game results and the results are globally identical among clients.

The disadvantages of C/S systems compared to P2P systems are longer network latency, slower responsiveness, and more resource hungry. Network latency is the time elapsed from the time of data transmission to the time of data arrival. On the one hand, the data packet from one client must travel through the server to be propagated to destined clients in C/S systems. On the other hand, packets are directly transferred to destination nodes in P2P systems. Even its own packet travels to the server and returns to the origin node to be projected in C/S systems. This implies that

responsiveness is worse in C/S systems than that in P2P systems. Responsiveness is the time taken for users from the occurrence of input to the projection of output. Furthermore, the server may be a bottleneck because it handles all the inputs from clients. Therefore, the server requires strong process power and large amount of storage capability (Pellegrino and Dovrolis, 2003).

A Distributed Model

Each peer manages its objects and communicates with other peers to synchronise the status of the objects in P2P systems. Therefore, this model can achieve very high responsiveness. Also, network latency is shorter than that of C/S systems on account of removing the necessity of transferring data to a server. This may improve game execution speed because peers do not need to wait for response from the server. However, peers may have unsynchronised status information due to the lacking of central synchronisation mechanisms and network latency which longer than frame interval.

The advantages of P2P systems compared to C/S systems are robustness, minimum delays, and lower requirement of hardware infrastructure. Different from C/S systems, P2P systems does not have single point failure. As mentioned before, messages from each peer are not delayed by the server. Also, each peer maintains its own database for object status and performs necessary cycling procedures such as input, calculation, and output. Therefore, P2P systems may not be suitable for game sessions which contain the large number of players. However, P2P systems work well for game sessions with small number of players and the hardware requirement of each peer is much lower than that of the server (Gossweiler, 1994; Bauer et al., 2002).

A Hybrid Model

To overcome aforementioned weakness of C/S and P2P systems, many researchers suggested hybrid systems. One of them is Funkhouser (1995)'s system called RING. RING adopts multiple servers and each server maintains multiple clients. Each client sends its data to the its server and then the server relays the data to the other servers. This system is a mixture of C/S and P2P systems. Clients are connected to one server and servers in this system utilise a distributed network communication model. In the RING system, servers can filter, alter, and insert messages. The filtering system is closely related to the visibility of objects in each client. Figure 2.14 shows the communication model of the RING system.

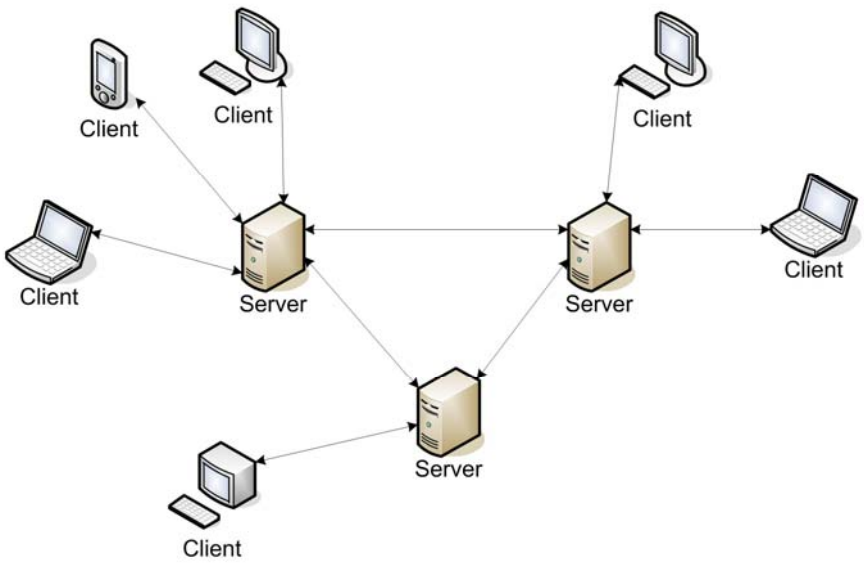


Figure 2.14: The RING communication model

The advantages of the RING system are reduced number of objects to process in clients, alleviated client processing requirement, and the efficient usage of network. In the RING system, each client processes only visible objects. Therefore, the number of

objects to store and process in the client sides is not directly related to total number of objects in the session.

Furthermore, the filtering occurs in the servers and this reduces the processing requirement of clients. To alleviate bandwidth requirement and increase the efficiency of network usage, server are connected to each other through high-bandwidth and low latency network. Therefore, clients may not need to connect to the server through high-bandwidth network (They may still need to connect to server which has low latency to increase playability).

One disadvantage of the RING system is extra network latency. The relay system among servers and extra calculations in servers prolong network latency.

Das et al. (1997) developed a virtual community system for children from 7 to 11 years old called HistoryCity. This system can support about 500 concurrent players using distributed servers and one master server which is called NetEffect architecture shown in Figure 2.15.

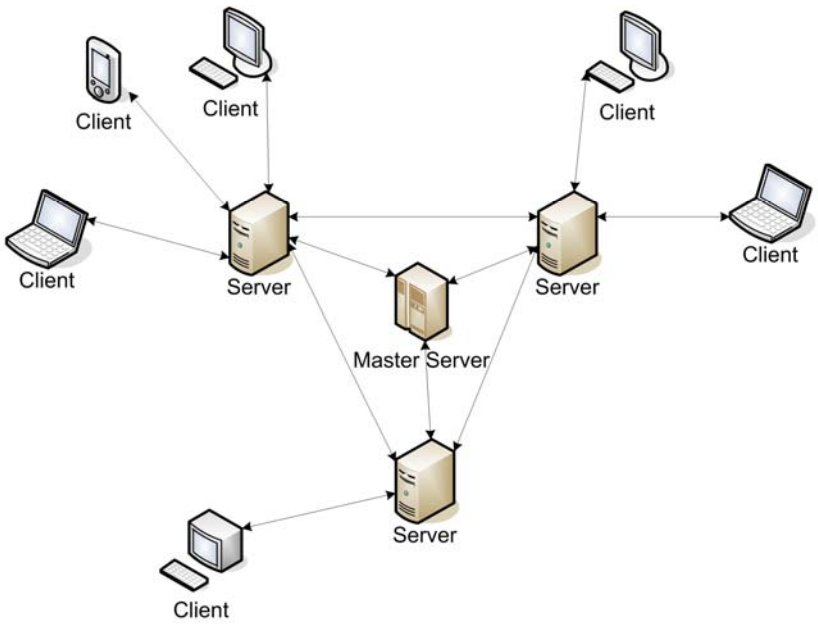


Figure 2.15: Overview of NetEffect architecture.

As shown in the figure, clients are connected to one server and the servers use a P2P network communication model. These servers are called peer servers (PSs) and the PSs communicate with a master server (MS) in the form of C/S model. Different from the RING system, the clients in this system can change their server depending on network and session situation. Clients, PSs, and MS propagate their packets using Transmission Control Protocol (TCP). One of the roles of MS is a load balancing of PSs. When PSs suffer from processing overloading due to crowded players, MS dictates clients to change their servers to uncongested ones.

The advantages of the NetEffect architecture are high scalability, and reduced bandwidth requirement. Multiple servers are utilised to manage clients and a load balancing occurs when needed by the master server. Bandwidth requirement of client sides is lowered by the utilisation of dead-reckoning (DR) algorithms. The DR algorithms and other consistency maintenance algorithms are explained in following sections.

The disadvantages of this system are extra latency and single point of failure. Because this system relays messages from clients through at least one server, network latency is prolonged. When the number of total players exceeds the processing limitation of the master server, network latency can be further delayed due to extra time for searching database. In worse case, the system can be failed when the master server crashes due to overcrowded players.

MiMaze is a 3D packman-like game which utilises unreliable protocols for exchanging game related data. The purpose of this game is destroying other avatars in the labyrinthine environment as many as possible (Gautier & Diot, 1998). Avatar is a

2D or 3D characteristic representation of players in NVE, CVE, and game applications (Bickmore et al., 1998). Figure 2.16 depicts the architecture of the MiMae.

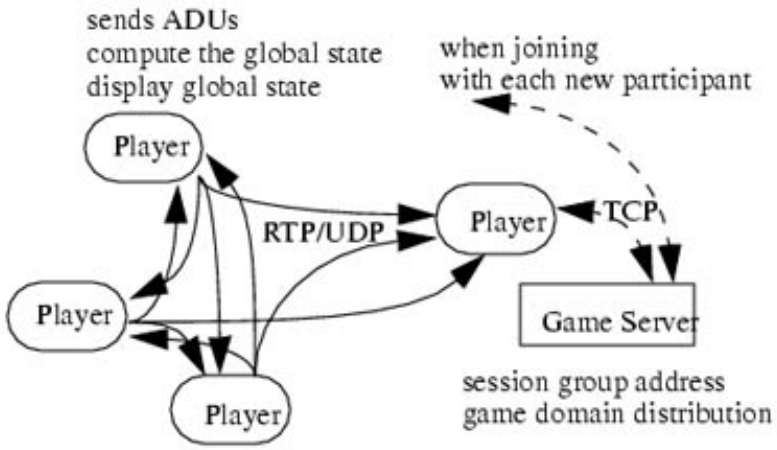


Figure 2.16: MiMaze Architecture (Diot & Gautier, 1999)

The difference between P2P systems and MiMaze is the existence of the game server as shown in the figure above. The game server provides current game session information to connected players using TCP. When a game starts, each player communicates with each other utilising unreliable multicast protocol, RTP/UDP. The characteristics of MiMaze are reduced bandwidth requirement and the adoption of a network latency masking algorithm, Bucket Synchronisation Algorithm (BSA). By utilising a multicast protocol, RTP/UTP, the bandwidth requirement of player nodes is reduced significantly. To maintain consistency and mask network latency a consistency maintenance algorithm, BSA, is used. The explanation on the BSA is in following sections.

2.3. Consistency Maintenance

As mentioned previous sections, distributed network communication models require consistency maintenance algorithms to maintain the consistency of object status

among participants in game sessions. In this section, various consistency maintenance algorithms are explained.

2.3.1. Overview

One of main aims of MODIGs is to provide an environment where realistic interactions among players can occur. For that, every participant should not realise any inconsistency of object status. In C/S systems, the server maintains the object state consistency therefore, the task is much simpler compared with that of P2P systems. As aforementioned, however, C/S systems have several disadvantages such as single-point-of-failure, longer network latency, and slower response.

To overcome these weaknesses, the MODIGs application designers may prefer replicated distributed network communication models (Vogel & Mauve, 2001). The term "replicated distributed" implies that the synchronisation mechanisms of shared object status may be required. These shared object status are called dynamic shared state (Singhal & Zyda, 1999).

In distributed communication models, each participant maintains the status of objects in the form of replicated objects if the participant does not possess the ownership of the objects. This type of management is called remote modelling and the replicated objects are called shadow objects or replicas (Singhal, 1996). Due to multiple concurrent updates which are not ordered properly, each replica in participants may contain inconsistent status (Bhola et al., 1998). The inconsistency of replicas may cause confusedness and/or irritation to participants. Therefore, it is an important task to maintain consistence the status of replicas in distributed network systems.

In general, consistency maintenance algorithms are categorised into two groups, pessimistic and optimistic approaches. These approaches are presented in following sections.

2.3.2. Pessimistic Approaches

Pessimistic approaches use lock or flow control based methods to maintain the consistency of the status of replicas in each participant (Munson & Dewan, 1996). Lock based approaches are similar to token ring type network which allows data transmission for a node that has a token in the network. One participant is allowed for the manipulation of shared objects only if the participant obtains a lock for the action exclusively in the lock based approaches. Even though these approaches provide simple and strong means to maintain consistency, transferring the ownership or lock of shared objects can be delayed due to network latency (Pantel & Wolf, 2002).

Sarin and Greif (1985) adopted flow control for preventing the confliction of multiple concurrent operations among participants. Their application allows only one participant for interaction with targeted objects. For that reason, this kind of approaches is limited to the applications of small number of participants.

DistView developed by Prakash and Shim (1994) is another example of a networked interactive application that utilises lock based consistency maintenance algorithms. In DistView, user actions on targeted objects are not permitted until the ownership or lock is obtained. DIVE (Hagsand, 1996), SPLINE (Waters et al., 1997), CAVERNsoft (Leigh et al., 1997), Virtual Society (Lea et al., 1997), and Bricknet (Singh et al., 1994) also utilise pessimistic approaches for maintaining the consistency of the status of shared objects.

GroupKit (Greenbug & Marwood, 1994) is also an application toolkit that uses lock based approaches for replicas synchronisation. However, this toolkit allows user actions on targeted objects while the participant obtains the lock for the objects. When the obtaining of the lock fails the pre-performed actions are cancelled and the states of objects are rolled back to its original status (Munson & Dewan, 1996).

In CIAO (Sung et al., 1999), only one action is allowed during the waiting time for obtaining the ownership or lock of intended shared objects. In the same manner as the GroupKit, roll back procedure happens when the procedure of obtaining the ownership or lock fails. This approach improves system performance by allowing the temporary inconsistency of state of shared objects.

Lee et al. (2001) propose prediction based consistency maintenance algorithms which can increase real time interactivity by reducing the necessity of communications for ownership passing among participants. This system utilises ownership prediction based on the distance between objects and participants. When the ownership of shared objects is predicted the participant transfers the request of the ownership to multiple nodes using multicasting protocols to reduce bandwidth requirement.

2.3.3. Optimistic Approaches

Optimistic consistency maintenance approaches are proposed to maximise real time interactivity among participant by removing the necessity of waiting for the ownership obtainment of shared objects or authorisation from central servers.

These approaches allow immediate update of the state of shared objects in the participant's local database and transmit her/his commands or the results to other participants. This implies immediate responsiveness disregarding the situation of

network such as delayed network latency due to network congestion. However, when inconsistency of the state of shared objects is detected the inconsistency should be fixed for maintaining consistency. The state repair procedure may cause confusion and irritation to participants. Furthermore, the procedure may harm the efficiency of system performance due to command re-ordering and re-processing (Bhola et al., 1998). Frequent State Regeneration (FSR), Dead Reckoning (DR), Bucket Synchronisation (BS), and Time Warp (TR) algorithms belong to the optimistic consistency maintenance approaches.

Frequent State Regeneration (FSR)

Certain types of MODIGs do not require absolute consistency during game sessions especially when network latency is relatively short. These types emphasise the smooth movement of shared objects than the precision of the location of them (Singhal & Zyda, 1999). Generally, these types of MODIGs do not make any presumption on the state of replicas to minimise the divergence between the state of real objects and that of replicas. Frequent State Regeneration (FSR) algorithm is designed for this kind of situation. The FSR algorithm sends packets which contain complete data regarding current frames in game sessions disregarding the occurrence of state changes at regular interval. Therefore, recipient sides can maintain consistency by repairing the divergence between the states of real objects and that of replicas which managed by transmitters even several packets are dropped during the transmission.

Dogfight (SGI, 2005), a flight simulation game from Silicon Graphics, utilises the FSR algorithm and transmits packets using broadcasting protocols. These packets are used to render air-crafts on the screen of participants in game sessions. Doom (Doom

World, 2003), Diablo (Blizzard, 2005), and Interstate '76 (Wikipedia - Interstate '76, 2006) employ the FSR algorithm for consistency maintenance.

The FSR algorithm is relatively easy to implement and does not require central servers nor lock management. However, this algorithm impairs bandwidth requirement due to packet transmission at regular interval. This gets worse when the number of objects increases in game sessions because packets need to contain complete data regarding the state of shared objects. Another weakness of this algorithm is that the degree of inconsistency may exceed the threshold of certain type of applications easily when network congestion occurs (Singhal & Zyda, 1999). Apparently, consistency sensitive applications such as co-operational medical applications may be not suitable for the adoption of the FSR algorithm.

Dead Reckoning (DR)

Originally, DR algorithm is used for finding the location of ships in navigation when any alternative method fails. It is known that DR is short for deduced reckoning (The Straight Dope, 2002). For deducing the location of ships, current speed and direction of ships are used with the ships' previously estimated or measured positions.

DIS (Distributed Interactive Simulation) which developed by DARPA (Defense Advanced Research Projects Agency) is a standard for distributed wargame platform and it adopts DR algorithms (DIS, 1993). The DR algorithm sends packets when divergence between real objects and ghost objects (replicas) exceeds pre-fixed threshold.

For example, node A manages spaceship 1 and its location (3, 3), speed (3 pixels per second), and orientation (0 degree: upward) are known to every participants in the

game session. If the player who controls spaceship 1 does not manipulate the state of the spaceship1 then the location of space 1 becomes (6, 3) 1 second later. Remote players utilise the DR algorithm to calculate the location of space 1 and render the spaceship on their screen. In this case, the node A does not need to transmit packets regarding the space 1.

However, if manipulation on the space 1 happens in node A and the result causes divergence that is larger than the threshold then appropriate packets must propagated to minimise inconsistency between real and ghost objects. Therefore, the DR algorithm can reduce bandwidth requirement significantly compared to the FSR algorithm.

The threshold is an important parameter in the DR algorithm. The smaller value of threshold is, the more frequent transmission of update packets and precise estimated state of ghost objects are (Cai et al., 1999). Prediction and convergence techniques are two other elements that required in the DR algorithm. Prediction is a technique to deduce current state based on previously received update packets. One-step and multi-step formulas can be used for the prediction. One-step formulas use only the last update packet and extrapolate the state of the object. Multi-step formulas utilise more than two update packets for the extrapolation. Figure 2.17 depicts the two types of extrapolation equations. In the figure, *x, v, a,* and *t* represent location, velocity, acceleration, and elapsed time since the last update respectively.

	One-Step	*Two-Step*
1^{st} Order	$x_t = x_{t'} + v_{t'}\tau$	$x_t = x_{t'} + \frac{x_{t'} - x_{t''}}{t' - t''}\tau$
2^{nd} Order	$x_t = x_{t'} + v_{t'}\tau + 0.5a_{t'}\tau^2$	$x_t = x_{t'} + v_{t'}\tau + 0.5\frac{v_{t'} - v_{t''}}{t' - t''}\tau^2$

Figure 2.17: Extrapolation Equations (Cai et al., 1999)

The convergence technique is used for resolving the divergence of real and ghost objects. The simplest method is moving the objects to the newly calculated location directly. However, this method can cause confusion to players because objects may warp or jump to new locations unexpectedly. More realistic convergence technique moves divergent objects smoothly to new location from current position through newly calculated path. The path is generated based on current velocity and orientation of the objects not to make unrealistic sudden movement.

The DR algorithm can reduce bandwidth requirement, however, it has several limitations. First, the DR algorithm permits temporary inconsistency due to the usage of threshold. Therefore, the state of shared objects can be unsynchronised until update packets which contains correct data for the real object arrive. Mauve (2000) explains how dead man can shoot when the DR algorithm is adopted in his paper. Second, the DR algorithm is more complicated to implement, maintain, and evaluate compared to the FSR algorithm. Furthermore, prediction for the location and orientation of shared objects may not be precise so, distributed agreement protocols may be required for collision detection. Third, the DR algorithm may not work well with fast action type MODIGs that rapid and sudden movement occurs frequently due to real time user interactions.

Bucket Synchronisation Algorithm

As aforementioned, the DR algorithm immediately commit the action of local users and reduce the state of remote objects which controlled by other participants. This method may trigger off inconsistency between real and ghost objects due to the improperly ordered and/or imprecise processing of commands from local and remote participants. Furthermore, the convergence procedure may create unexpected artefacts

such as the jumping or warping of objects during game sessions (Vogel & Mauve, 2001).

MiMaze (Diot & Gautier, 1999), a 3D packman-like distributed network multiplayer game, adopts Bucket Synchronisation (BS) algorithm and Vogel and Mauve (2001) propose local lag which is similar to the BS algorithm. Figure 2.18 shows how the BS algorithm works. As shown in the figure, game time is divided into the block of predefined length and each block contains one bucket. When packets from remote users are delivered to each local user the packets are stored into appropriate buckets. The assignment of packets into buckets depends on prefixed playout delay. In other words, the playout delay determines how much time must be passed to display the result of user actions. Actions from local users are also stored into buckets and the calculations of finding appropriate buckets for the commands are performed using the playout delay.

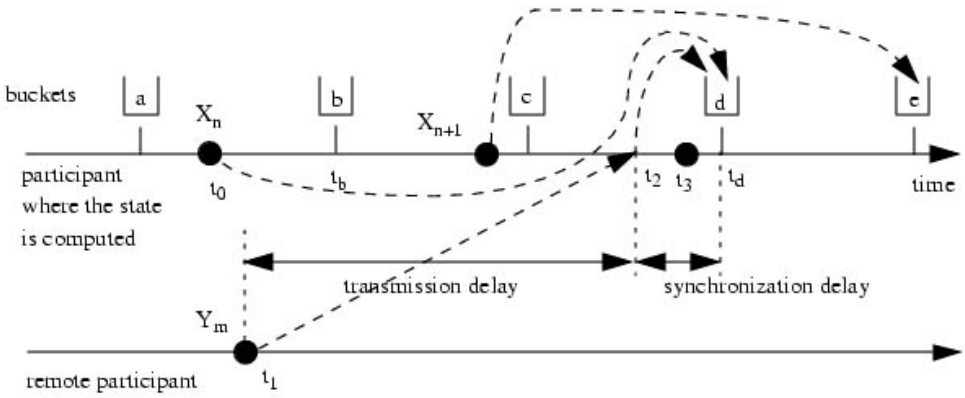


Figure 2.18: The bucket synchronisation mechanism

The data from stored packets are used to render game frames in sequential order. When packets from remote players are dropped or delayed, packets in the preceding bucket are utilised as they are or for the deduction of the state of objects when the DR

algorithm is exploited with the BS algorithm. The BS algorithm is able to resolve short-term inconsistence, however, players may detect slow responsiveness due to the playout delay or local lag (Vogel & Mauve, 2001).

Time Warp

Time Warp (TW) is an optimistic consistency maintenance algorithm that adopts virtual time paradigm (Jefferson, 1985). The virtual time paradigm denotes the utilisation of one dimensional temporary coordination system for the management of distributed processing. The virtual time follows the clock conditions which are proposed by Lamport (1978).

The two fundamental semantic rules are (Jefferson & Sowizral, 1982):

- **Rule 1:**.The virtual departure time of messages must be smaller than the virtual reception time of the messages.

- **Rule 2:**.The virtual time of current event must be smaller than the virtual time of following events.

The above rules show that the virtual time of events and data transmissions always increases positively. Also, the propagation delays of event messages are always positive values. The time warp algorithm processes messages from local and remote participants if the values of timestamp (virtual time) of the messages are larger than that of the latest processed event because the global ordering of events are correct at the currently known situation. However, the TW algorithm must perform a rollback procedure when a message with smaller timestamp than the latest one arrives to maintain the global ordering of event processing.

The rollback procedure consists of two mechanisms. First is cancelling previously transferred messages which are not ordered in sequential temporary arrangement. Second is ordering messages properly and processing the ordered messages again. For the cancelling messages, the TW algorithm sends anti-messages and these anti-messages may cause the rollback of other processes (Damitio et al., 1994).

The rollback procedure may trigger the explosion of the transmissions of anti-messages that may cause the severe degrading of system performance. Furthermore, sending anti-messages requires storing the negative messages of previously transmitted messages that consumes system resources (Lubachevsky, 1989; Steinman, 1993).

Vogel and Mauve (2001) addressed the disadvantage of the TW algorithm that maintaining the global ordering of events results in the consumption of significant process power and resources. Also, implementing the TW algorithm in networked applications increases the complication of the application logics.

2.4. Resource Management

Network bandwidth and process power are limited resources that significantly affect the scalability of MODIGs. Singhal and Zyda (1999) define the relationship among network resources, timeliness, and system performance that can be summarised as the below formula.

Resource ≈ M x H x B x T x P where,

- **M:** number of messages transmitted in the MODIGs
- **H:** average number of destination hosts for each message

- **B:** average amount of network bandwidth required for a message to each destination
- **T:** timeliness with which the network must deliver packets to each destination
- **P:** number of processor cycles required to receive and process each message

Increasing the efficiency of one variable can degrade that of other variables. For example, the efficiency of variable B can be improved when packet compression techniques are used by reducing average packet size. However, the efficiency of variable P drops due to encoding and decoding procedures. Therefore, which variable's efficiency should be improved should be carefully determined according to the requirements and resource bottlenecks of MODIGs (Smed et al., 2002).

To optimise the utilisation of these resources various techniques such as interest management, packet compression, and packet aggregation are proposed. In following sections, these techniques are explained in detail.

2.4.1. Interest Management

Interest management techniques reduce the number of packets to be transferred to other participants who are interested in the packets only by utilising filtering mechanisms such as application specific and/or network attribute specific. The most well known network attribute specific filtering mechanism is IP multicast protocol which explained in the previous section. Application specific filtering mechanism is also called as intrinsic filters due to its characteristic of the determination on packet transmission based on application specific data content (Smed et al., 2002).

RING (Funkhouser, 1996) utilises distributed servers which filter messages from participants depend on potential visibility of objects. Greenhalgh and Benford (1997) introduce the concept of awareness in distributed network applications. The awareness quantifies the significance of a target object to other objects in remote nodes and two terms, focus and nimbus are defined for the measurement. The focus indicates the interest of observers on certain medium and the nimbus represents the expected areas of the observable on the given medium. When the focus and nimbus (called as aura in all) are coincided with each other, the observer can receive the information in regards to the observable as it shown in Figure 2.19.

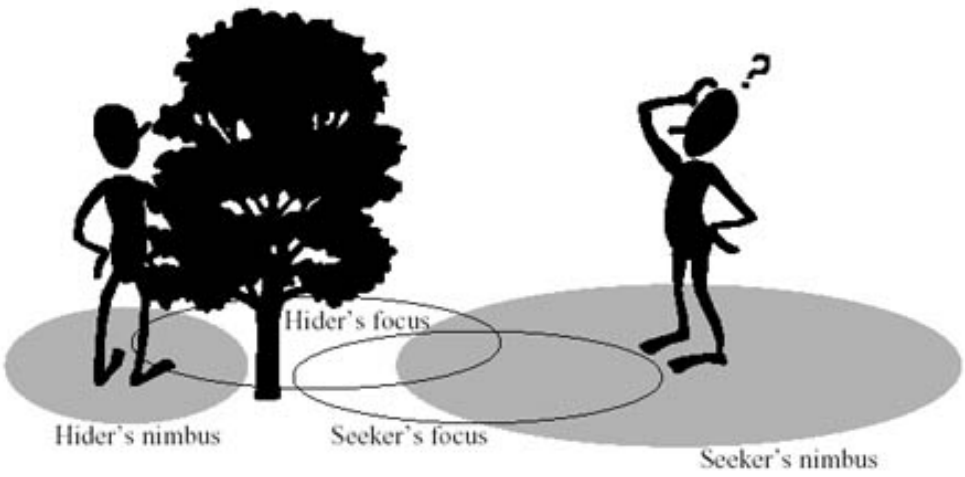


Figure 2.19: Focus and Nimbus (Greenhalgh & Benford, 1997)

MASSIVE-1 system which is implemented by Greenhalgh and Benford (1997) utilises three media types and these have different awareness criterion. The collision of focus and nimbus is detected by an aura collision manager. When the collision is detected the aura collision manager signals two nodes and then peer connection is established between the two nodes for data transmission.

NPSNET system (Macedonia et al., 1995) separates its virtual environment into hexagonal cells and filters messages using the regional division. In this system, update information on objects is transferred to remote participants who controls objects in the same region. Spline system (Barrus et al., 1996) also divides its virtual environment into regions similar to the NPSNET system but the difference between them is that the Spline system uses arbitrary shape for the division. Rectangular grid regions, hexagonal regions, or arbitrary shapes are adopted as the shape of regional division in various NVE applications (Russo et al., 1995; Mastaglio & Challahan, 1996; Macedonia et al., 1995; Barrus et al., 1996). The interest management techniques which use regional division methods are also applied in MODIGs especially for Massively Multiplayer Online Role Playing Games (MMORPG) due to the simplicity of implementation and the necessity of the alleviation on the burden of managing the large number of concurrent players.

2.4.2. Packet Compression

Packet compression techniques can reduce average packet size by encoding packet header and/or body to be transferred to other nodes (Tye & Fairhurst, 2003). In general, compression methods can be divided into two groups, lossless and lossy compressions (Kim, 2000). On the one hand, lossless compression methods can restore original data from encoded one without any loss. These methods are usually utilised for text documents and executable files because these types of data may not be usable any longer if they can not be recovered into their original state from the compressed one. On the other hand, lossy compression algorithms are not required to recover the original state of compressed data. These algorithms are broadly utilised to compress multimedia data significantly compared to the lossless compression

algorithms by exploiting the imperfection of human perceptibility such as visibility and audibility.

Packet compression methods can be categorised into internal and external types (Singhal & Zyda, 1999). Internal compression algorithms encode packets without referring previously transferred packets. The advantages of this type of methods are the easiness of implementation and no requirement of reliable protocols. However, this type does have a low compression rate compared to external compression methods. Different from the internal compression algorithms, high compression rate can be achieved with external compression methods by utilising packets transmitted beforehand. However, the external compression methods require memory space for storing packet history. Furthermore, reliable transmission protocols are demanded for satisfy the dependency of packets.

2.4.3. Packet Aggregation

The bandwidth requirement can be alleviated by utilizing packet aggregation techniques. The packet aggregation techniques reduce the overhead of packet headers by combining multiple packets into one packet. For example, the usual header size of UDP protocol is 28 bytes and that of TCP is 40 bytes (Forouzan, 2006). When timeliness is not a major concern on data transmission, an aggregated packet which contains single header and multiple user commands or states data can be delivered to target nodes. Consequently, the alleviated bandwidth requirement may improve system performance by reducing the number of headers to be processed.

The packet aggregation methods can be categorised into timeout and quorum-based types (Singhal & Zyda, 1999). On the one hand, timeout-based packet aggregation (or

forced delay aggregation) methods collect and propagate packets which generated in configured timeout period (Grunwald et al., 2006). This type of methods guarantees the upper bound of packet transmission delay. The efficiency of the time-based algorithms depends on the frequency of packet generation during the timeout period. On the other hand, packets are aggregated until arranged numbers of packets are produced in the case of quorum-based packet aggregation methods. Therefore, this kind of packet aggregation methods assures the reduction rate of bandwidth requirement but does not limit packet transmission delay.

To address the weaknesses of timeout and quorum-based packet aggregation methods, hybrid algorithms can be utilized that adopt timer and counter to determine packet transmission. When timeout occurs or sufficient packets are collected, the aggregated packet is propagated to other participants and the counter and timer are reset. In general, aforementioned packet aggregation methods sacrifice timeliness for the alleviation of bandwidth requirement. In chapter 6, the quorum-based packet aggregation method is explained with experimental results.

3. Research Methodology

This research is focused on improvement of network latency and bandwidth while maintaining consistency between players in multiplayer online digital games (MODIGs). To actualise the aims which also mentioned in chapter 1, several approaches and algorithms are proposed and these are explained in following sections. Also, the procedure of node selection and data collection will be justified in appropriate sections.

3.1. Research Questions

The main research questions in this thesis are:

- Is it possible to actualise multiplayer online digital games (MODIGs) under the condition of heterogeneous and unpredictable latency network environment such as the Internet?

- If it is not possible then what are the major factors that hampering the actualisation of this type of applications?

- What approaches can be adopted to rectify and/or improve the problems?

Various literatures (Bernier, 2001; Bettner & Terrano, 2001; Blow, 1998; Cheshire, 1996; Diot & Gautier, 1999; Fritsch et al., 2005; Jiang and Boustead, 2005; Mauve, 2000) show that two network factors, latency and bandwidth, are causing difficulties on the actualisation of MODIGs. Diverse approaches are proposed to resolve the main two problems and actualise MODIGs on the Internet (Bangun & Beadle, 1997; Bauer et al., 2002; Baughman, & Levine, 2001; Beigbeder et al., 2004; Blow, 1998; Cronin

et al., 2004; Ng, 1997; Pantel & Wolf, 2002). Therefore, several sub research questions are determined. These are:

- What are the characteristics, advantages, and disadvantages of previous and current approaches to get around the two major problems such as latency and bandwidth in creating playable MODIGs?

- What are the main reasons for maintaining state consistency among players and the advantages and disadvantages of various consistency maintenance algorithms?

- What factors would affect the playability of MODIGS? Would it improve the efficiency of MODIGs in terms of game execution speed by the sacrifice of resources such as process and bandwidth to rectify latency related problems?

After the realisation of significant problems on the actualisation of MODIGs and the literature review process on various approaches for solving the problems, several approaches are proposed mainly to solve network latency problem. These approaches also have induced several research questions such as:

- How would locked bucket synchronisation (LBS) algorithm improve network latency problem without sacrificing the playability of MODIGs? Would the LBS algorithm performs equally well on the simulation environment, COMP2P, and the MODIG such as Duel-X?

- Would tree-based P2P approach be able to reduce network latency and/or bandwidth requirement for each player? How could the approach ensure robustness and enable the functionality of any-time join (joining game sessions which are on playing)?

- What is the relationship between bandwidth requirement and packet drop rate in the case of the adoption of various transmission schemes? How effective are the transmission schemes on reducing network latency and what are the side effects of the schemes?

- How would packet aggregation help to reduce bandwidth requirement of each player? Would it delay packet propagation and degrade the efficiency of MODIGs in terms of game execution speed by delaying packet transfer?

These research questions will be examined and answered in appropriate chapters that explain proposed approaches in detail.

3.2. Hypothesis

Various hypotheses are made from literature reviews and preliminary experimental results. These are validated in corresponding chapters with the experimental results from COMP2P and Duel-X. These hypotheses are:

- Frequent State Regeneration (FSR) algorithm will achieve almost optimal game execution speed measured by fps (frame per second) and bandwidth requirement measured by outbound PPS (packets per second) in any case of experimental conditions compared with Lockstep (LS) and Locked Bucket Synchronisation (LBS) algorithms.

- PPS (inbound) in FSR will depend on PDR (packet drop rate) during game sessions.

- Game execution speed in LS (lockstep) algorithm is directly related to latency and the latency will be affected by packet drop and delay during game sessions.

- PPS (outbound and inbound) in LS & LBS algorithms will be directly affected by pre-measured Round Trip Time (RTT) values, packet drop rate, and packet delay.

- LBS algorithm can mask latency by sacrificing responsiveness and the responsiveness is directly connected to playout delay (elapsed time for the presenting results from the occurrence of the user input). Therefore, the longer playout delay the higher fps in case of WAN environment.

- When latency is smaller than frame interval, it would not cause severe harm on fps in any case of the aforementioned three synchronisation algorithms, FSR, LS, and LBS.

3.3. Limitation and Needs for Innovative Solutions

3.3.1. Consistency problems

There are at least three basic operations a MODIG main engine loop has to perform: it must read input from the user, move objects in the world (including the viewpoint) based on internal simulation (which is influenced by that input), and draw the current state of the world to the screen. A typical MODIG application main engine loop may look like below (Below, 1998):

```
while (1) {
```

```
        read_input ( );       // keyboard, joystick, network packets or whatever;

        move_objects ( );   // change object movement parameters based on input

        draw_screen ( );    // all the graphics

    }
```

The computer screens of participants are updated by changing the video buffers in their video cards. The video buffer is also called the "frame buffer". The application screen updating speed is measured by frame rate, which represents how many frames are being drawn per one second.

For example, 30 fps (frames per second) means the video card updates the frame buffer 30 times per second. Therefore, at 30 fps it takes about 33ms to draw 1 frame. If a MODIG application's frame rate is 30 fps then each node should send its packets in 33ms to eliminate the necessity for consistency protocols.

Figure 3.1 shows the out of synchronisation problem. Operation O1 that is generated at site1 is transmitted to site2 and site3. O2 from site2 and O3 from site3 are also passed to site1, site2, and site3. If O1, O2, and O3 are propagated to neighbouring nodes in time (in this case, before frame 2) then no consistency protocol needed. However, as it can be seen, O1 is passed to O3 after frame 3 and before frame 4. Therefore, the user at site3 may make incorrect decision due to the lack of information in regard to the operation O1 in between frame 2 and 3.

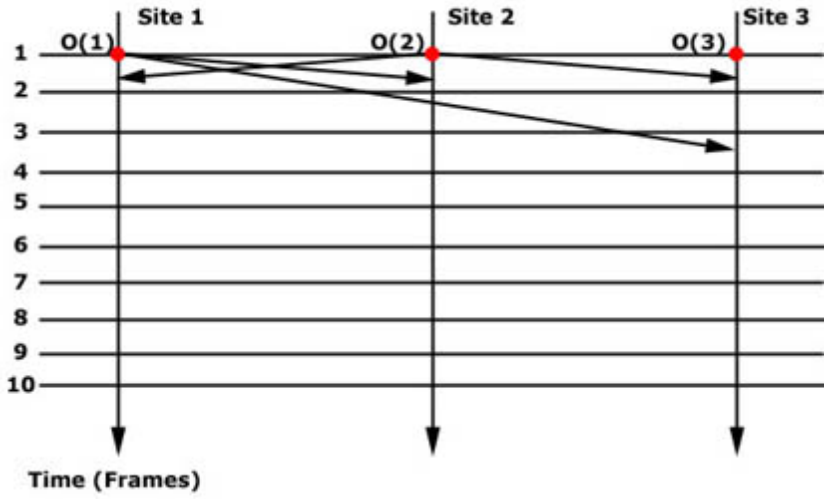


Figure 3.1: Out of Synchronisation

Therefore, inconsistency is unavoidable in MODIGs due to the network latency problems. The inconsistency problems can be categorised into three sub groups such as divergence, causal violation, and intention violation (Chen, 2001).

Divergence can be divided into two: intra object divergence and inter object divergence. **Intra object divergence** occurs when two or more sites execute different operation on same object. As the result of such operations, multiple versions of same object are generated. Those operations are called conflict operation. For example, robot 1 controlled by node 1 may move a ball into position (10, 5) and transmit the operation to other nodes in an online robot soccer simulator. At the same time or before the packet arrives, robot 2 in node 2 can also move the ball into position (5, 10). The result of this conflicting operation is inconsistent (multiple version of) dynamic shared state in each node.

Inter object divergence is the result of operations that manipulate different objects but those objects are placed at overlapping locations. For example, different users can construct their buildings at the same place in real time strategy (RTS) games if no

consistency protocol is provided. In online robot soccer simulators, multiple objects such as robots and a ball can occupy same location when the inter object divergence occurs.

Even though the operation 1 (O1) in node 1 happens earlier than the operation 2 (O2) in node 2, the O2 may arrive earlier than O1 at node 3. It can be problematic or erratic if O2 refers to O1. For example, O1 is an operation that creates mine at position (10, 10) and O2 is move operation that makes unit1 move position (11, 11) from (9, 9). Other nodes such as node 1 and node 2 may see detonation of unit 1 but node3 may not see that because O2 arrived earlier than O1. If this MODIG application applies the dead reckoning algorithm then unit 1 may be marching until the detonation or dead acknowledgement packet arrives. The user of node 3 will be confused or frustrated when sees the sudden death of unit 1 by the unreasonable detonation.

The final state may be only one when the divergence happens and the deleted or overwritten intentions of users are ignored. For example, the user of node 1 changes the purpose of building 1 to defence and at the same time, the user of node 2 fixes the purpose of the same building to offence. If the choice of node 1 user wins then the intention of user 2 is ignored and user 1 may not know user 2's intention at all.

Chen (2001) uses object replication to handle intention violation due to simultaneous operations changing the same object. As the result, multiple versions of the same object are created and displayed. The participants can either select a certain version or keep them all. Drawbacks to this approach are a complex management of multiple versions of the same object, and a confusion effect for the user if too many different versions exist. Furthermore the approach seems to be problematic to apply on

MODIGs. The object duplication would result in two representations of the same object in the game and it would give users confusion (Vogel & Martine, 2002).

3.3.2. Network Latency

The speed of light in vacuum is approximately 300,000 kilometres per second. However, when travelling through fibre, light travels at roughly 66% of its speed through vacuum, so the rate is only about 200,000 kilometres per second. The Earth's circumference is approximately 40,076 kilometres. Therefore, it would take around 100 ms to travel to the opposite location from current position (only one way).

Stuart (1996) has measured the packet travel time from Stanford to Boston, which the distance is about 4,320km. The speed of light in fibre is approximately 200,000 km as it is mentioned before. Therefore, estimated travel time is around 21.6ms (4,320 km / 200,000 km) and round trip time to Boston and back is about 43.2ms. The result ping time from Stanford to Boston over the Internet is about 85ms. It is within a factor of two of the theoretical optimum. The additional delay (around 42ms) comes from the nodes and the network itself. Obviously it takes time for the data to travel through the computer's operating system and network hardware even before it reaches the network. Routers and modems introduce their own delays to the data.

With Ethernet local area network, latency is typically under 10ms. If you are dialling into the Internet by a modem through the telephone system, your latency will be at least 100ms. Transcontinental transfers require an additional 60-150ms, and intercontinental latencies can range from 250-500ms or more (Singhal & Zyda, 1999). Therefore, generally, it may be difficult to implement highly interactive MODIGs without proper consistency protocol in a distributed manner.

3.3.3. Reliability vs. Unreliability

Certain consistency protocols assume the use of reliable network protocol in its MODIG to support reliability in terms of transmission guarantee and message ordering. However, because of the reliability and ordering semantics, the reliable protocol must transmit more information and introduce more network latency.

For example, the sender or receiver must check whether the sent packets are dropped or not during transmission. When packet dropping or corruption happens then retransmit must go though to complete the reliable transmission. Packet ordering mechanism requires similar process. Therefore, it may increase overall system efficiency to manipulate unreliable protocols to the direction of increasing reliability and of keeping the protocol overhead.

3.3.4. Existing Approaches

As discussed in preceding sections, inconsistency happens due to network latency and consistency protocols are implemented to solve divergence and causal violation problems. The most well known optimistic algorithm, **Time Warp**, maintains consistency when causal ordering violation happens. Roll over procedure occurs to fix the inconsistency by sending anti-messages to other nodes and redo operations that are re-ordered. Its main drawback is that it needs to go back to the time that inconsistency happened and do the operations again. It may cause confusion and frustration to participants. Furthermore, it assumes the usage of reliable network protocols that generate network overhead. It is normally not recommended to use reliable network protocol for non-time-sensitive (trivial) data because the trivial data tends to be overwritten by following same objects' data. Finally, it increases total run

time because it goes back to past and go through the processes that occurred before when it catches inconsistency between hosts.

Frequent state regeneration mechanism normally uses unreliable network protocol and transmits objects' properties at every frame. Some packets may be dropped or delayed but arrived packets keep overall consistency. This method increases total bandwidth but its advantage is the simplicity of the mechanism. However, it does not contain fix mechanism for divergence and causal violation.

Bucket synchronisation method does not execute the issued operation immediately but delays the operation until the current time is same as next bucket time. It considers network lag time and collects the operations from other nodes and itself into buckets, which are connected to certain time or frames to utilise the fact that generally less than 100ms local lag is not distinguishable to human brain (Gautier et al., 2001). This method is also, easy to implement but there is no mechanism for divergence and causal violation. Also, Global clock mechanism such as Network Time Protocol (NTP) is required to maintain time synchronisation among nodes in the MODIGs.

3.4. Outline of Research Methods

1970s and 1980s were the golden age of arcade games and the dominant genre of games in these days was fast action games such as spacewar, Space Invaders, Asteroids, Galaxian, Missile Command, Galaga, Xevious, and so on (Burnham, 2001). After the debut of the Internet and the development of networking technology, the new era of networking games has begun. However, it is still a difficult task to actualise fast-paced action MODIGs due to the unconquerable physical limitation, the transfer speed of electronic and optical signal, under current technology.

First, thoughtful literature reviewing process has been occurred:

- To realise current situation of the development and significance of MODIGs
- To broaden knowledge on the current limitation and problems on the actualisation of MODIGs

Second, various preliminary experiments have carried out such as RTT measurement among multiple nodes which reside different countries, tracing routes, drop rate measurement at various hours (off-peak and on-peak hours), playability test on responsive time, and so on.

Third, several approaches are proposed and COMP2P is implemented for the verification and validation of these approaches. Experiments, data gathering and analysis of the results occur for the verification and validation.

Forth, a fast-paced action MODIG, Duel-X, is implemented to verify the efficiency of the proposed approaches on the Internet.

3.5. Proposed Approaches

To alleviate the overhead of managing consistency between real and ghost objects in each player, four approaches are proposed: **Locked Bucket Synchronisation (LBS) algorithm, Tree-based P2P system, Smart Transmission Scheme (STS), and packet aggregation**. Each approach will be explained in this section.

3.5.1. Overview

The main aims of this research can be divided into two groups. The first one is reducing network latency between players. The second one is to decrease bandwidth requirements for each player.

The former is one of the most difficult tasks to actualise due to the physical restriction of data transfer speed. To alleviate this problem, Locked Bucket Synchronisation (LBS) algorithm is proposed and tree-based P2P system is utilised with this algorithm. To prevent network delay caused by packet drop, Smart Transmission Scheme (STS) is proposed.

For the aforementioned second aim, alleviating bandwidth requirement, packet aggregation method is proposed and it can be achieved in tree-based P2P system. The detailed explanation for the approaches mentioned above will be given in following sections.

3.5.2. Consistency Maintenance Algorithms

Two conservative consistency maintenance algorithms are implemented and these algorithms are briefly explained here. The full explanation will be given in chapter 5. Due to incomparable characteristics between conservative and optimistic consistency maintenance algorithms, the comparisons between them are not presented in this thesis. Also, experimental results on optimistic algorithms are omitted because various research papers have shown the experimental results on these algorithms. However, the quantity of experimental results on conservative algorithms is relatively small especially in online gaming area. Therefore, full size experiments have been executed and the results are shown in appropriate chapters.

3.5.2.1. Lockstep Algorithm

The lockstep algorithm (LS) is one of the simplest algorithms in consistency maintenance algorithms. It simply waits its process until the process is assured to be safe to move forward. Therefore, it keeps its status perfectly synchronised with other nodes' one. However, this characteristic significantly reduces its performance.

To improve its performance slightly, the LS algorithm which implemented in this thesis does not wait acknowledgement packets for its sending packets. Instead, it utilises opponents' packets as acknowledgement because of the characteristics of conservative algorithms, which is safe process can be executed. The safeness of the process can be confirmed by counting the number of arrived packets for each frame on the implemented LS algorithm in this thesis.

3.5.2.2. Locked Bucket Synchronization Algorithm

The improvement achieved by removing wait-for-acknowledgement is not significant in the LS algorithm. Experimental results on LS algorithm show that this algorithm is not appropriate for long latency in network games. To avoid jerky movement, the frame speed should be higher than 10 FPS (Gautier et al., 2001) and this implies that the network latency longer than 100ms may affect badly on the playability of network games when LS algorithm is applied.

To overcome this problem, bucket synchronisation algorithm (Diot & Gautier, 1999) is introduced and the experimental results have shown significant improvement compare to LS algorithm. However, this algorithm does not accompany with conservative consistency maintenance but does with optimistic algorithm, which is dead reckoning algorithm. One of the aims of this research is maintaining perfect

consistency between players to prevent confusedness and irritation of players, Locked Bucket Synchronisation (LBS) algorithm is proposed in this thesis.

This algorithm is similar to the LS algorithm in terms of assurance of safeness for process execution. However, it introduces the concept of playout delay from the aforementioned bucket synchronisation algorithm. Therefore, it is a hybrid system of the two algorithms and it is capable of masking network latency without harming consistency among players. Full explanation and charts for this algorithm is provided in chapter 5.

3.5.3. Tree-based P2P System

The Internet was developed to increase possibility of successful communication among military bases in nucleor war by introducing excessive paths between networks. The interconnectivity of networks brings multiple paths between nodes and a router is one of devices that in responsible for managing the information of paths in the Internet. Various path finding algorithms are introduced such as Routing Information Protocol (RIP), Open Shortest Path First (OSPF), Border Gateway Protocol (BGP), and so on to find the shortest path between origin and destination nodes (Forouzan, 2006).

However, there are countless networks which are connected to the Internet and it is simply not possible that routers can hold direct connections to all of the other routers. Therefore, when direct connection is not possible then routers must find alternative indirect paths to the destination routers and/or networks.

This fact may imply that sometimes it is faster to transfer data to target nodes if indirect paths are used. In other words, the shortest path exists between node A and B

which is not direct. For example, travelling through path A-C-B takes shorter time than through path A-B. To find these alternative indirect paths, tree-base P2P system is proposed. There are other reasons and cases that indirect paths are preferred to direct paths and these are fully explained in chapter 6.

3.5.4. Smart Transmission Scheme

Packet drop is not easily predictable and it is one of major causes of delaying packet transferring. When packet drops it can be detected by negative acknowledgement scheme, which is analogous to time-out system. Acknowledgement packets are supposed to be arrived in time and if not, packet resending occurs in case of packet drop.

In case of packet drop, it can be a data packet from origin or an acknowledgement packet from destination. If it is the latter then it may not cause process execution delay due to the characteristics of implied LS and LBS algorithms. However, the former case significantly delays the execution of process because the other opponents never transfer their packets until the dropped packet arrives to their network buffer. Therefore, the process will be halt until other opponents packets arrive.

To prevent this kind of delay, Smart Transmission Scheme (STS) is proposed. This scheme judges the importance of sending packets by calculating predicted arrival time of the packets and if the packets are significant then identical packets with them are sent immediately after the departure of these packets. In short, same packets are sent twice in case of packet drop. This scheme is compared with two other schemes namely, normal and blind transmission schemes for efficiency comparison. Full explanation and experimental results are displayed in chapter 6.

3.5.5. Packet Aggregation

When blind and smart transmission schemes are adopted in networked games, the bandwidth requirement of each player can be worse compare to the case of adoption of normal transmission scheme. To reduce this side effect, packet aggregation method is proposed in this thesis.

The packet aggregation method exploits the characteristic of conservative consistency maintenance algorithms that is these algorithms wait for the safeness of process execution. Due to this attribute, the process speed of game sessions follows the speed of the slowest node.

On other side, this implies that some node may have additional time which can be used for reducing bandwidth requirement. The additional time earned is exploited for packet aggregation. When tree-based P2P system is utilized packets need to be relayed and these packets can be aggregated together into one packet.

Each packet has common packet header information such as target IP address, port number, total length, and so on in transport level. Also there is additional information in application level such as target node numbers, packet type, frame number, and so on. These savings can alleviate bandwidth requirement without increasing network latency. Detailed explanation and experimental results are provided in chapter 6.

3.6. Node Selection

Total eight nodes are selected for networking experiments which utilise a real time online shooting game, Duel-X and chapter 4 explains the implementation of this game in detail. Generally, most current P2P-based networking games support maximum

eight players due to the limitation of network resources. One more obvious reason for the relatively small number of nodes is due to the difficulty of organising a large number of remote nodes.

Before the implementation of Duel-X, a compact P2P based network simulator (COMP2P) has been designed and implemented for overcoming limited time and resources. In simulation world, there is no limitation except computer memory and CPU process power. To be realistic, however, three groups of nodes are created such as four, eight, and sixteen. The latencies between these nodes are randomly generated between 5 to 100 ms for the clearness of performance analysis. According to our preliminary experiments, direct path is usually the shortest path between nodes reside in different networks. If there is an indirect path which is shorter than direct path, its difference is usually smaller than 10% of the network latency of the direct path.

However, it should be fully examined in assigning realistic latency values among nodes for network experiments which use the COMP2P in future research.

3.7. Data Collection

Network latency and bandwidth related experimental data is collected and displayed in appropriate chapters. The major sources for these values are the experimental data from COMP2P and Duel-X. The combinations of proposed approaches generate unique data sets and appropriate charts are presented in each chapter for the approaches. Figure 3.2 displays the combinations of each approach in detail.

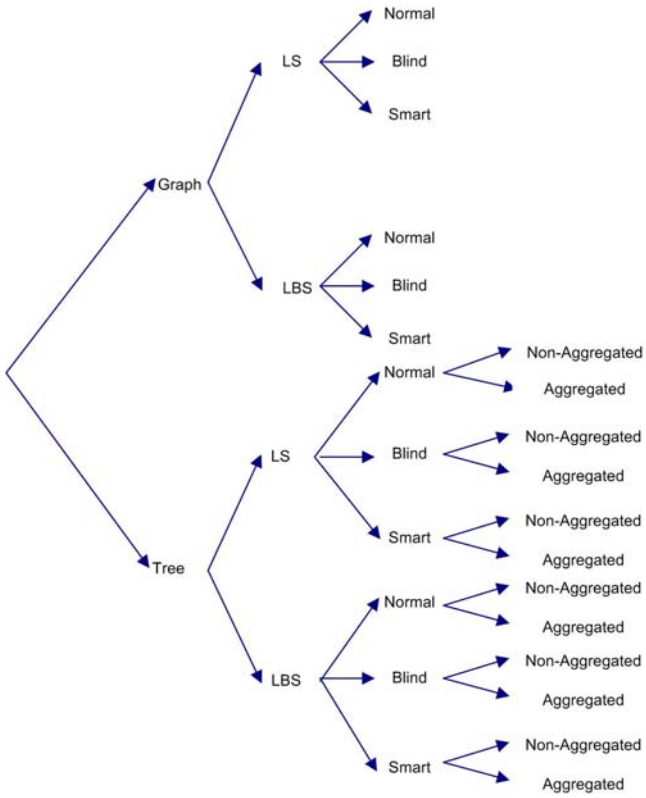


Figure 3.2: Combinations of proposed approaches

Network latency related data fields in experimental results are packet departure time, packet arrival time, resending frequency, and RTT time (acknowledgement packet arrival time – game packet departure time). Bandwidth related data fields are total number of packets sent, received, resent, and dropped. These data values are collected from all the possible combinations using two experimental instruments, which will be explained in following section.

3.8. Instruments

Two testing beds, COMP2P and Duel-X, are built for the analysis of the performance of proposed approaches. Experiments on networking are difficult tasks due to the unpredictability of network environment and behaviour. To be efficient on resource management, a compact P2P based network simulator (COMP2P) is implemented and

various experiments are executed to verify the performance of algorithms and approaches which will be fully examined in following chapters.

3.8.1. System Overview

After the experiments on COMP2P, an online arcade game, Duel-X, is developed for the assurance of the adoptability of the proposed approaches on real world applications. Also, a number of experiments are processed on this testing bed using several combinations of approaches for the verification.

Common parts of their architecture are explained here, however, full explanation of each application will be presented in next chapter. Figure 3.3 displays the system architecture of both applications.

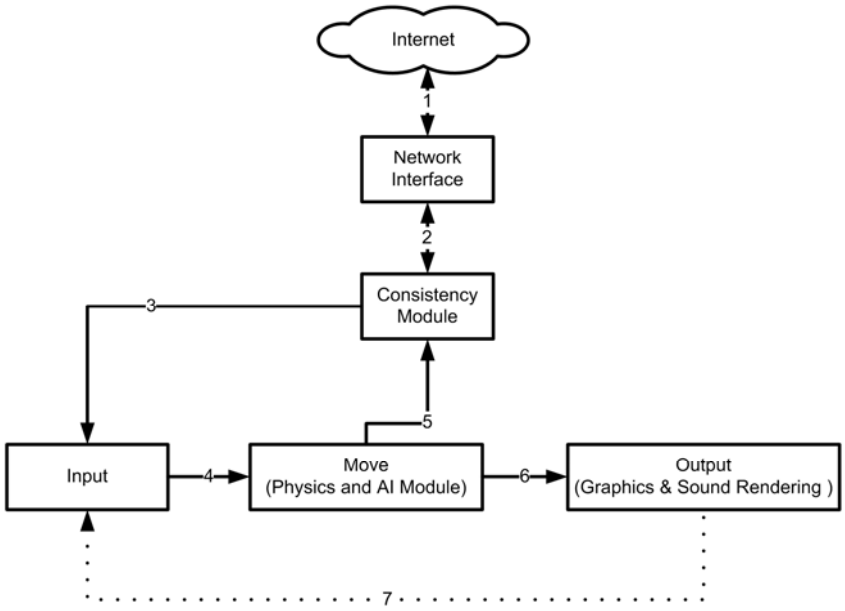


Figure 3.3: The System Architecture of the COMP2P and duel-x.

As seen in the figure, there are five modules such as network interface, consistency module, input, move, and output module. The general application process model (Input – Move – Output) is used and the explanation for those modules is omitted.

The network interface and consistency module are explained in following sections. The aims, assumptions, and parameters in this system are:

Aims:

- Playable responsiveness is one of the main concerns
- Reducing network latency without effecting consistency among players.
- Alleviating bandwidth requirement of each player and super nodes (or parent nodes) without harming overall system performance.

Assumptions:

- Discrete and distributed (peer-to-peer and frame based) system
- Utilisation of unreliable and reliable network protocols
- Multiple nodes (multi-user system)
- Bandwidth requirement is not a main concern but network latency is because the network latency related problems are hard to tackle but bandwidth related problems have many suggested solutions compare to the other one.

Parameters:

- Frame interval
- Playout delay
- Transmission schemes

- Packet aggregation types
- Game duration
- Number of nodes (for COMP2P only)

3.8.2. Network Module

As shown in Figure 3.4, the reliable and unreliable network protocols are used and all the packets received are recorded in the packet management table for future reference.

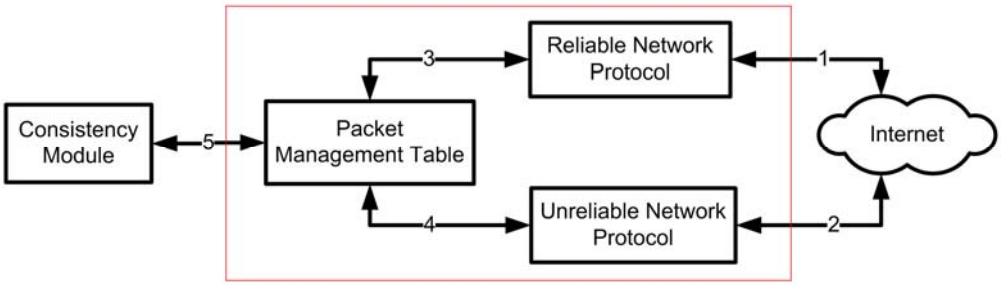


Figure 3.4: The Network Module

Lobby operations are transmitted through reliable network protocol and game operations are sent by unreliable network protocol. Lobby operations are mainly related to communication between lobby system and players. Due to the characteristics of these operations, high reliability and low urgency, these operations will be carried on a reliable network protocol such as Transmission Control Protocol (TCP). **Game operations** are that operations related to game processing and these are game data such as players' command or status of objects in the game, acknowledgement packets, and so on. To propagate these operations without any delay, a low overhead unreliable network protocol such as UDP is utilised.

When the Frequent State Regeneration (FSR) algorithm is applied, the usage of unreliable network protocol is not a concern because arrived same objects' state packets solve short-term inconsistency. However, the scheme overflows network bandwidth with objects' state packets.

To ensure the arrival of game operations in other cases of consistency maintenance algorithms, a **negative acknowledgement scheme** is used. It is that whenever events happen, the unique numbers of received packets are sent to other nodes. When the acknowledgements are arrived at other nodes then corresponding node-packet number table is updated so, each node is acknowledged properly and if missing or dropped packet is detected then packet resending occurs.

3.8.3. Consistency Module

Multiple consistency maintenance algorithms are implemented for performance comparisons between them. The first algorithm is frequent state regeneration algorithm and as explained in previous chapters it does not pay attention on the situation of network environment and simply transmit current frame information to other players. Synchronisation only happens when packets arrive in time and late packets are simply discarded.

Two conservative consistency maintenance algorithms are applied in the consistency module. LS and LBS algorithms are very similar to each other and in fact, LS algorithm is identical to LBS except the difference of playout delay between them. The playout delay of LS is same as its frame interval and that of LBS is fixed by the system when game sessions start. Aforementioned algorithms, FSR, LS, and LBS, manipulate the state of objects and the command table as shown in Figure 3.5. The

modified table information is read by input module and used to update players' state. Full explanation will be given in following chapter.

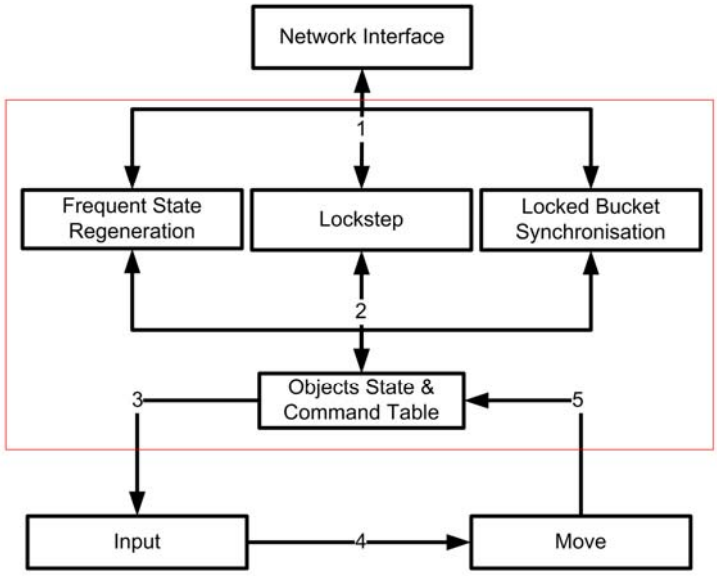


Figure 3.5: The Consistency Module

4. Experimental Systems

In this study, a number of approaches and algorithms are proposed, with a view to alleviating latency and reducing bandwidth requirements in order to improve the performance of distributed network games. While the logic of the proposed innovations is often quite clear intuitively, it remains essential to demonstrate the feasibility, applicability and efficiency of each proposal in turn. One way to achieve this, as well as to provide an opportunity to further develop and finetune the proposed ideas, is to implement each innovation in the context of real-world game sessions, or of a simulated game environment. This study adopts both of these approaches to the experimentation/implementation tasks, and the current chapter describes how they are operationalised.

Applying new approaches to real-world domain applications is a time-consuming task, especially where the network environment is uncertain, and the actual experimentation requires complex interactions between multiple players, parameters and variables. A useful first step, therefore, would be to implement a network simulator which is flexible enough, and realistic enough, to allow wide-ranging explorations of the system being analysed and to help gain a broad understanding of the main patterns, issues, and insights involved. In section 4.1, such a network simulator is described. In terms of programming and software development, the work reported in this section represents a major component of the research completed for this study. The simulator, called COMP2P, will be used to test the effects of a range of innovations, particularly those reported in chapter 6 below.

Ultimately, the most satisfactory (and challenging) approach to experimentation is to work with a real-world environment. In this research, a real-world application, Duel-X, which is an online arcade game, is implemented to verify that the results from the simulator are applicable to the real world domain. To conserve resources, this approach is applied mainly to a subset of the proposed innovations, namely those discussed in chapter 5. Section 4.2 below provides further information about Duel-X and how it is utilised in this study. Section 4.3 provides a summary of the main points raised in this chapter.

4.1. Network Simulator

The existence of the multi-factors makes it difficult to analyse the performance of algorithms in networking area, such as unpredictable latency, uncertainty of packet drop, and irregular CPU processing condition, etc. To remove irrelevant variables from networking experiment, a compact P2P (COMP2P) network simulator is implemented, which is flexible enough to experiment the proposed approaches on various P2P systems. A network simulator creates virtual environment that mimics the behaviours of network.

Alternative solution for the performance analysis of these approaches involved the usage of existing network simulators such as NIST (Carson & Santay, 2003), NS-2 (Haddad & Gordon, 2002), cnet (McDonald, 1991), QualNet (Bagrodia et al., 2006), REAL (Keshav, 1997), etc. The main reason of using custom-made network simulator is driven from the fact that the approaches are operated in application level and based on the utilisation of UDP protocol rather than TCP. The most network simulators mentioned above are for the analysis of the dynamic behaviour of flow and congestion control schemes in packet-switched data networks (Bagrodia et al., 2006).

The adopted approaches in this research dynamically manipulate the topology of network and controls flows in application level by using UDP-based custom protocols. The final aim of implementing own network simulator is to give an easy understanding of the behaviour of the approaches.

4.1.1. System Overview

The COMP2P is implemented in C++ with standard template library (STL) and consists of three main classes, namely, player, simulator and statistics. Figure 4.1 shows the basic architecture of the COMP2P network simulator.

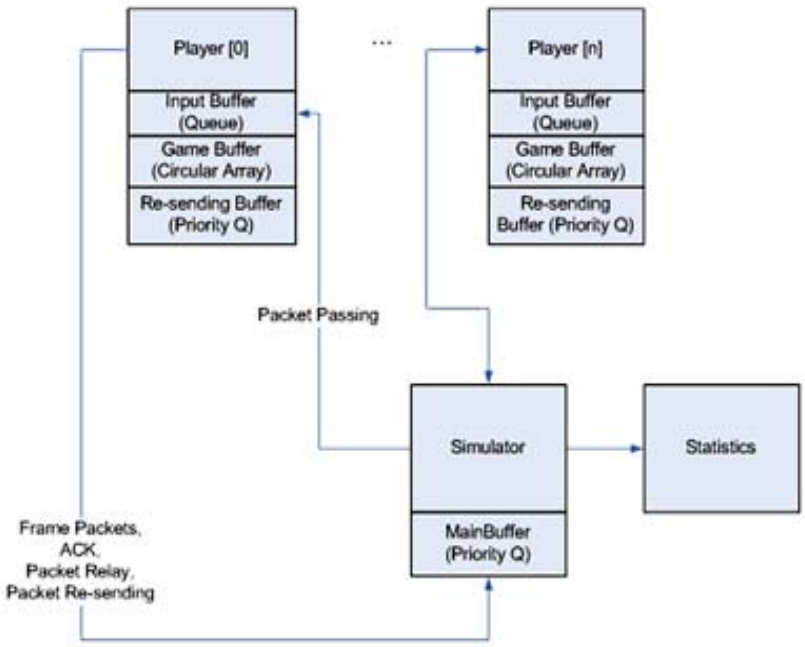


Figure 4.1: Tree-based P2P Network Simulator Architecture

When experiments start, network parameters are set up. These parameters are:

- The file name that contains node information (4, 8, and 16 nodes)
- The number of times of experiments

- Graph type (mesh or tree)
- Aggregation type (true or false)
- The type of consistency maintenance algorithms (frequent state regeneration, lockstep, and locked bucket synchronisation)
- Simulation duration in seconds
- Frame interval in milli-seconds
- Playout delay in milli-seconds
- Packet drop rate (from 0% to 10%)
- The type of transmission scheme (none, blind, and smart)

Then the simulator class controls overall process of the experiments. First, it passes appropriate parameters to the player class according to the parameters to initialise variables in the player class. After the initialisation, experiment time proceeds in the simulator class and it is passed to each player class. When the time is for packet generation then, each player checks the condition of packet generation depending on the parameters and the status of game buffer. The decision will be determined by the algorithms of the approaches that are chosen when the experiments start. These algorithms and approaches will be further explained specifically in following sections.

When the packet is generated then it is passed to the main buffer in the simulator class in the order of supposed packet arrival time. Therefore, priority queue is implemented for the main buffer to satisfy this condition. Even the packet is successfully generated it is not guaranteed to be delivered to the target nodes in real world situation. Since

packet drop is unpredictable, it may not be delivered to the destination. To simulate this behaviour, the binomial distribution of Poisson distribution (Papoulis, 1984) is used. Based on the discovery of Siméon-Denis Poisson, Equation 4.1 is used to decide whether the packet is deliverable to the destination or not. It is also determined in the simulator class. The explanation regarding the equation is out of scope of this thesis, so it is omitted.

$$P_p(n \mid N) = \frac{N!}{n!\,(N-n)!}\, p^n\, (1-p)^{N-n}. \qquad \text{(EQ. 4.1)}$$

Where N is the number of trials, n is the number of successes in N trials.

The simulator class accepts arrived packets and passes these packets to appropriate destinations according to the target addresses that are already recorded in the packets. The fields recorded in the packets are:

- Origin nodes list
- Destination nodes list
- Source packet id
- Unique packet id
- Frame number
- Source node
- Source departure time
- Departure time

- Arrival time
- Packet number
- Packet type
- Game Data
- Next packet

The structure of the packet would be simpler if class inheritance is utilised though. Due to time constrains for this study, the options which was not feasible or less relevant to the performance analysis are omitted from this research. For the implementation of Duel-x, this packet structure is improved because it is directly related to the network bandwidth issues.

When the packet is arrived at the destination node, the node checks whether the packet has to be relayed or not by reading the fields of the packet when tree-based P2P system is applied. If it is required to be relayed, then the intermediate nodes alter the values of fields appropriately and send it to the simulator class. The game data of arrived packets is stored in the game buffer of the player and used for game update. Every packet that is arrived at the simulator class will be passed to the statistics class for latter analysis. Figure 4.2 shows class diagram of COMP2P network simulator. The structure details of some classes are also beyond the research scope.

4.1.2. Simulator Class

The simulator class is responsible for packet forwarding, packet dropping, and passing simulation result to the statistics class. The main tasks of simulator class can be summarised as:

- Setting up experimental parameters and passing them to players
- Receiving packets from players and passing them to destination players and statistics class for future analysis
- Dropping packet determined by Poisson distribution
- Creating a weighted graph by reading from an external file which contains latency information between players

Shown in Figure 4.2, the simulator class contains appropriate access functions to set up and pass experimental parameters for the first task. For the convenience of experimental process, batch process is supported in the simulator class.

For the second task, a main buffer is provided to store packets from players and these packets are passed to appropriate destinations according to the field values in each packet. The main buffer is implemented by utilising a priority queue and the key factor of ordering packets is arrival time value in each packet. Experiment execution time is counted in one milli-second each and when the arrival time is bigger than current execution time then it is passed to the destination of the packet. Also, these packets are sent to statistics class for future analysis.

To actualise packet drop rate, Poisson distribution is adopted in the simulator class. When packet is to be passed to other player packet dropping is decided by the binomial functions which are calculated by the use of pre-determined packet drop rate.

4.1.3. Player Class

The player class utilizes three data structure types, Queue for an input buffer, Priority Queue for a re-sending buffer, and Circular Array for a game buffer to store other players' packets, to re-send dropped packets, and to gather frame data, respectively.

The main roles of player class can be enumerated as: packet generation, packet processing, packet relay, packet aggregation, packet de-aggregation, sending acknowledgement, and packet resending when packet drop is detected.

Experiment execution time is passed from the simulator class, and then if it is time for packet generation, then the player will generate packets to all other players. The decision will be made depending on the pre-determined approaches. The generated packets are sent to the simulator class and these packets are passed to the destinations as it is explained in previous sections. When the packets arrive at the target players, these packets are processed and stored in the game buffer of the players. Then each player checks whether the packet has to be relayed to other players if tree-based P2P system is adopted. Also, the aggregation of the arrived packets is verified. Then appropriate process follows packet de-aggregation or packet relay as determined by the situation and network parameters.

4.1.4. Statistics Class

The statistics class collects every packet that is sent to the main buffer of the simulator class for performance analysis by utilising the nodeInfo and framePacket classes, illustrated in Figure 4.2. The nodeInfo class is a collection of the framPackets class that stores packet information from the simulator class.

When experiments finish, three types of files are created, they are: experimental parameters, latency report, and experimental result. The first file stores all the experimental parameters and is used to distinguish from other files and keep the parameter information. The second file records time-related information in detail, for example, departure time, arrival time and relay time, etc. These output files are used to create latency related charts shown in following chapters. The last file keeps overall experimental results such as total dropped packets, total arrived packets, total packets, total latency for all, total latency for each frame, average latency for all, etc. Again, the information is applied on generating various graphs in the coming chapters.

4.1.5. Experimental Data Set

In general, most of network games support maximum eight players in P2P network due to network constraints such as latency and bandwidth. Therefore, it is decided to make the simulator initially with generated data sets of four, eight and sixteen players with randomly selected network latency with values between 5 and 100 ms.

Table 4.1, Table 4.2, and Table 4.3 show maximum and average latency between each node. The tables also present travel distance between each node in milliseconds

except Table 4.3. Maximum latency is a key factor which affects the overall game speed. The maximum latency of the four, eight, and sixteen-player experimental data set are 82, 77, and 100 respectively. The meaning of maximum latency will be explicated further in the next chapters.

Table 4.1: latency values between four players (Max Latency: 82, Average Latency: 46.83)

	0	1	2	3
0	0	5	59	23
1	5	0	82	61
2	59	82	0	51
3	23	61	51	0
Max:	**59**	**82**	**82**	**61**
Avg.:	**41**	**43**	**44.43**	**31.57**

Table 4.2: latency values between eight players(Max Latency: 77, Average Latency: 40.46)

	0	1	2	3	4	5	6	7
0	0	26	58	76	10	46	11	60
1	26	0	64	52	37	58	36	28
2	58	64	0	15	48	77	35	54
3	76	52	15	0	18	15	21	24
4	10	37	8	18	0	27	77	25
5	46	58	77	15	27	0	76	9
6	11	36	35	21	77	76	0	70
7	60	28	54	24	25	9	70	0
Max:	**76**	**64**	**77**	**76**	**77**	**77**	**77**	**70**
Avg.:	**41**	**43**	**44.43**	**31.57**	**34.57**	**44**	**46.57**	**38.57**

Table 4.3: latency values between sixteen players(Max Latency: 100, Average Latency: 52.10)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Max:	**91**	**99**	**89**	**100**	**96**	**93**	**96**	**100**	**91**	**96**	**97**	**89**	**96**	**89**	**100**	**100**
Avg.:	**55.87**	**32.80**	**52.47**	**60.93**	**42.40**	**60.07**	**55.67**	**48.47**	**47.13**	**59.67**	**51.27**	**41.67**	**54.07**	**50.20**	**58.13**	**62.80**

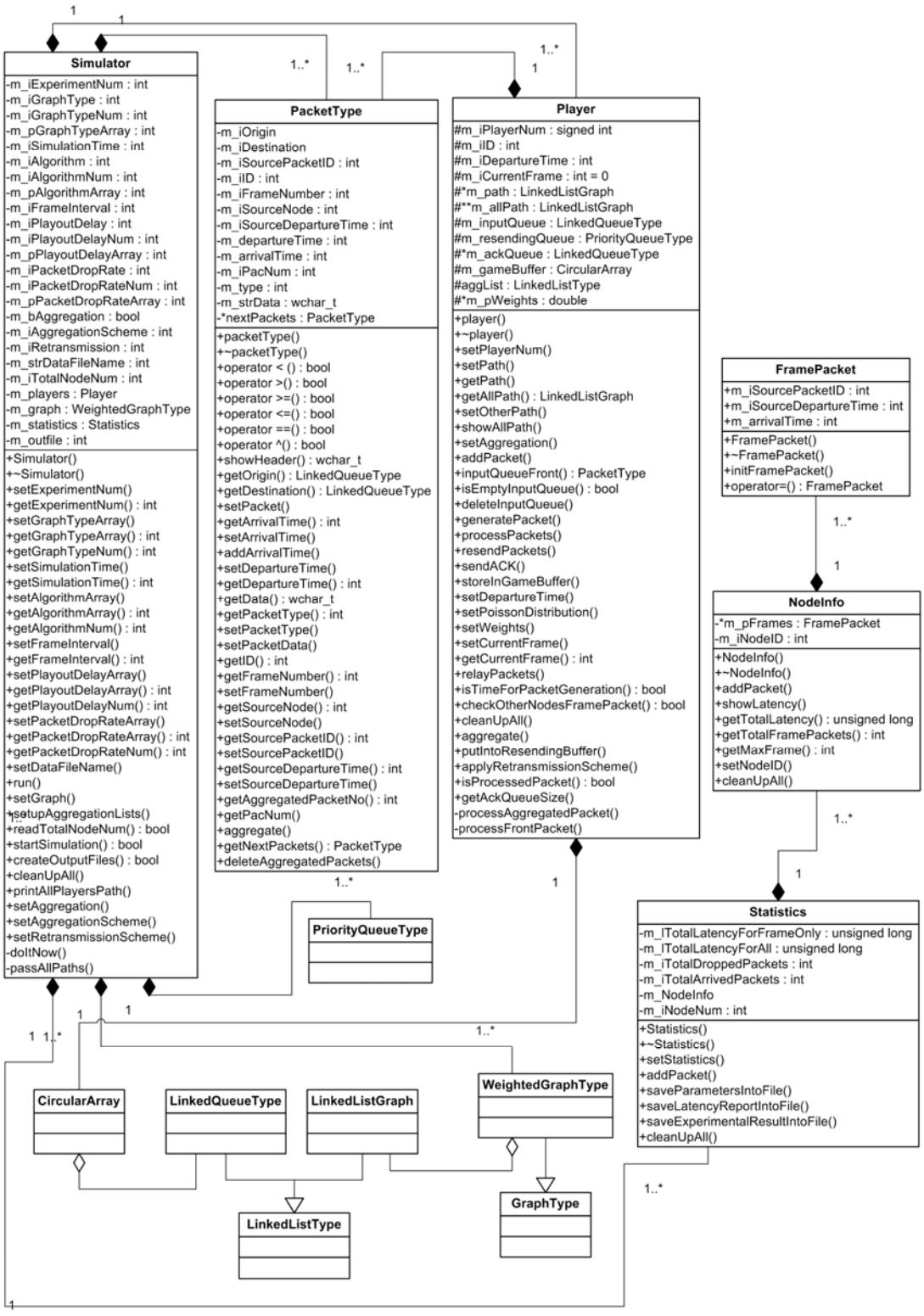


Figure 4.2: The class diagram of COMP2P network simulator

4.2. Duel-X

To verify the workability and efficiency of the proposed algorithms, and more importantly to do so in a real-world environment, some of these algorithms are implemented and tested in Duel-X, a real world application. Duel-X is an online shooting game which can be played on the Internet. Its main game-logic and codes are based on a game, called Duel, which is included in Microsoft's DirectX SDK to demonstrate the usage of DirectPlay, a part of the DirectX library for network communication.

In implementing Duel-X, all of DirectPlay parts are removed, and the game flows are modified significantly. In addition, various consistency algorithms are incorporated to allow the experimentation with the proposed approaches and the analysis of their effects of system performance.

4.2.1. System Overview

The Duel-x consists of four modules such as game, networking, lobby and web server modules. Each player manages its own and other players' status by using the game module. The game module transmits the data from the player by utilising the networking module. The lobby module is responsible for providing a meeting place for players and managing a game room list and providing the access information such as IP address and port number of the owner of each room. Finally, the web server provides services such as creating membership information, downloading applications, manuals, help, and so on. These four modules will be explained in detail in the following sections.

4.2.2. Web Server

The address of web server is http://www.int.gu.edu.au/~moon/duelx and five main sections are provided, such as news, message board, game download, hall of fame (rankings), and help. To play the game, Duel-x, players are required to gain authentification information such as user id and password. It can be done through membership functionality. Players can join, modify, withdraw membership, and after the creation of membership, then they can read, write, and modify messages on the forum. Also, they can download the game and patch through game downloading section. The help menu handles the functionality of manual, FAQ, questions and answers, etc. It also shows how to play this game. Players could also check their rankings through the rankings menu. Figure 4.3 shows message board section in the web server.

For membership and forum services, MS-SQL Server and PHP are utilized. The combination of Apache web server, HTML, Flash, and JavaScript is adopted to deliver the web contents related to this game. For a game launcher and game module upgrading systems, ActiveX was used to actualise the system at the initial stage. However, the strengthened security system of Windows XP service pack 2 does not allow the use of unsigned ActiveX component. Therefore, game launching and upgrading need to be performed manually by each player before they connect to the lobby module.

4.2.3. Lobby Module

The lobby module is an independent server for supplying a place where every player can connect to retrieve information in regards to a game room list and the network

related information of the creator of the games. The lobby application provides four services which are related to user information, player information, session information, and help.

The user information section provides an interface for searching, updating, adding, and deleting registered users. These functionalities are also accessible in the web server through membership service.

The player information section is for displaying currently logged players. This section also displays players' information such as ID, current IP address, win count, lose count, RTT measurement.

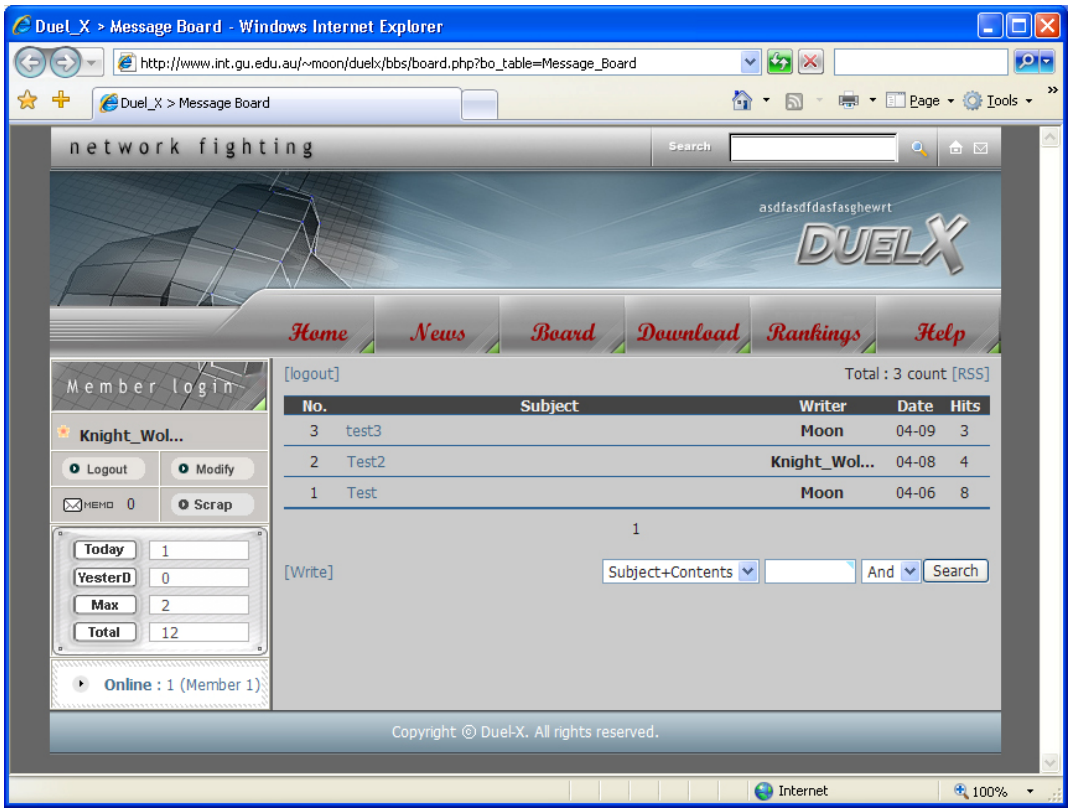


Figure 4.3: Message board section in web server

The session information section displays information regarding to currently available game session list. When players create a game room, then a new session begins and it is recorded in the lobby module.

Finally, the help section shows simple explanation about this module and how to use this application. Figure 4.4 shows the user information section in the lobby application.

To prevent connections from unauthorized players, the lobby server asks authentification information which is provided by the web server. When players connect to the lobby server, the transferred players' authentification information is validated by querying the information to MS-SQL Server. If the result is positive then the players can retrieve game room related information, such as name of the game, the information of the game room owner, etc.

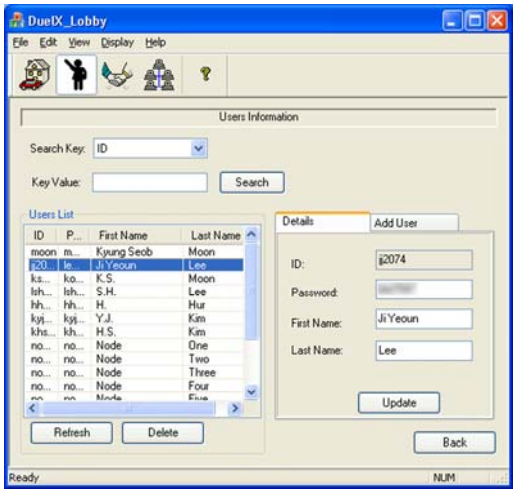


Figure 4.4: User information section in lobby module

The game availability test must be processed before the players decide to enter one of the game rooms. The purposes of the game availability testing are: 1) to test whether the game has been started already. 2) to verify the game owner is still online so, the

players can connect to the game room owner. 3) to examine the playability of the game, which means the RTT between each player is in boundary for this game. Currently there is no boundary of the RTT for this game because we are examining the playability of this game depending on various parameters such as network latency, bandwidth, playout delay, and so on. However, for actual network games, the examination is crucial to achieve the maximum playability of network games.

4.2.4. Game Module

The aim of this game, Duel-X is to destroy as many opponent ships as possible in limited time. For the purpose of game data collection, game duration is limited to one minute and auto-pilot function is turned on. As soon as the game is launched, a start up screen will show up. After the screen, the connection dialog screen will appear as it can be seen in Figure 4.5 and Figure 4.6.

Figure 4.5: Start up screen of the game, Duel-x

Players could log in the lobby server by entering their own authentification information, which was created through membership service in the Duel-X web server. If the lobby server allows the entrance of the players, then a main menu dialog window appears on their screen, shown in Figure 4.7.

Players now have a choice of creating a game room or joining existing game rooms. If first choice is creating a game room, then a game room creation dialog window appears to allow players to decide their game room name. Then they have to wait until other players join their game to start the session. These two dialog windows are shown in Figure 4.8 and Figure 4.9.

Figure 4.6: Connect to lobby dialog window of the game, Duel-x

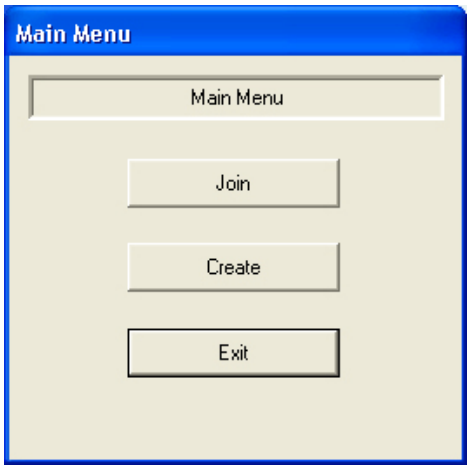


Figure 4.7: Main menu dialog window of the game, Duel-x

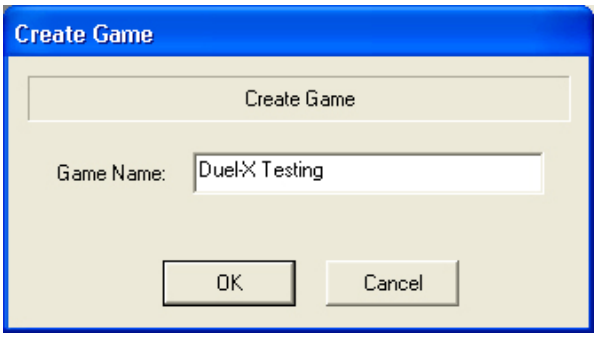


Figure 4.8: Create a game room dialog window of the game, Duel-x

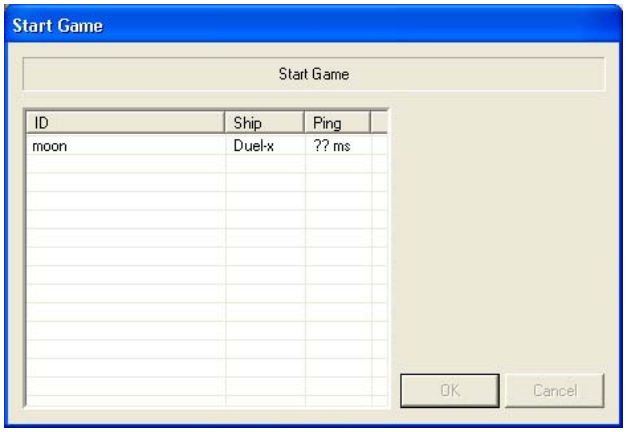


Figure 4.9: Start game session dialog window of the game, Duel-x

Joining a game room option shows a list of existing games sent by the lobby server. Players could choose one of the listed games and the request will be sent to the lobby

server and the owner of the game. If game availability test goes well then the joiner and owner will have direct connection between them. Figure 4.10 shows the joining a game room dialog window.

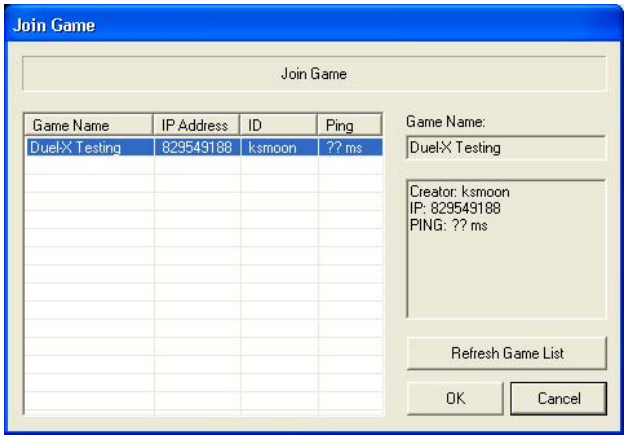


Figure 4.10: Joining game session dialog window of the game, Duel-x

Generally, the procedures up to this point only involve some tasks and functionalities in online games, especially for casual network games. However, Duel-X is a testing-bed for the performance analysis of the P2P network algorithms, therefore, it provides one more step to set up network parameters which can be seen in Figure 4.11. In this dialog window, experiments organiser allows the selection of P2P types, consistency maintenance algorithms, transmission scheme types, packet aggregation, experiment duration, frame interval, and playout delay. This window will only be available to the game owner side.

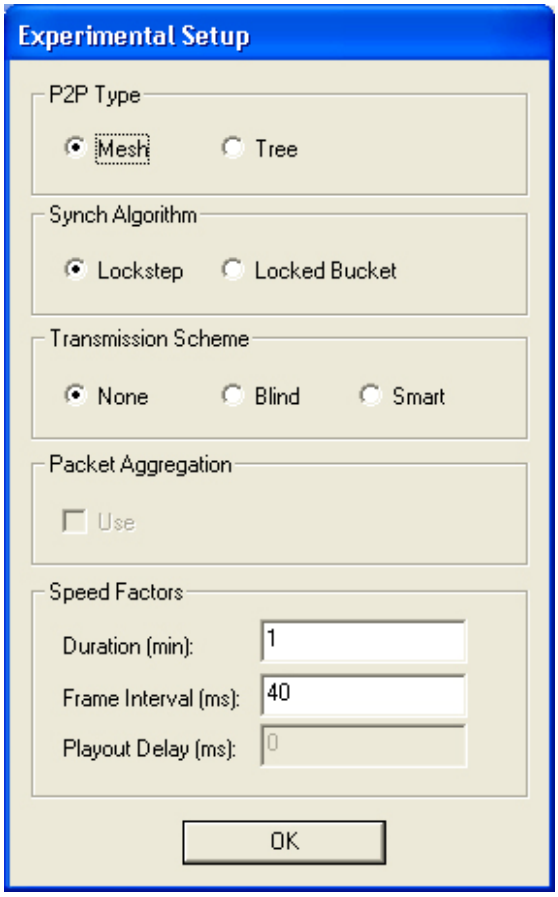


Figure 4.11: Experimental set-up dialog window of the game, Duel-x

After a game owner (the organiser of this experiment session) chooses all necessary network parameters, the game will finally start. In the game, each player can observe game related statistical information such as elapsed time, current frame, FPS, total packets, and so on. Figure 4.12 shows the game information dialog window.

At last, the game starts and Figure 4.13 shows game screen. The game is relatively simple compared to existing casual network games at commercial level. Not only time constraint of this PhD study but also clearness of performance analysis limits the development of complicated networking games. Because networking environment itself is unpredictable, it would be troublesome to analyse the performance of networking algorithms if CPU processing is also hard to predict. Therefore, network approaches will be concentrated on rather than the game logic itself. The screen

shows rendering speed on top of the screen and four space ships that chase to destroy each other.

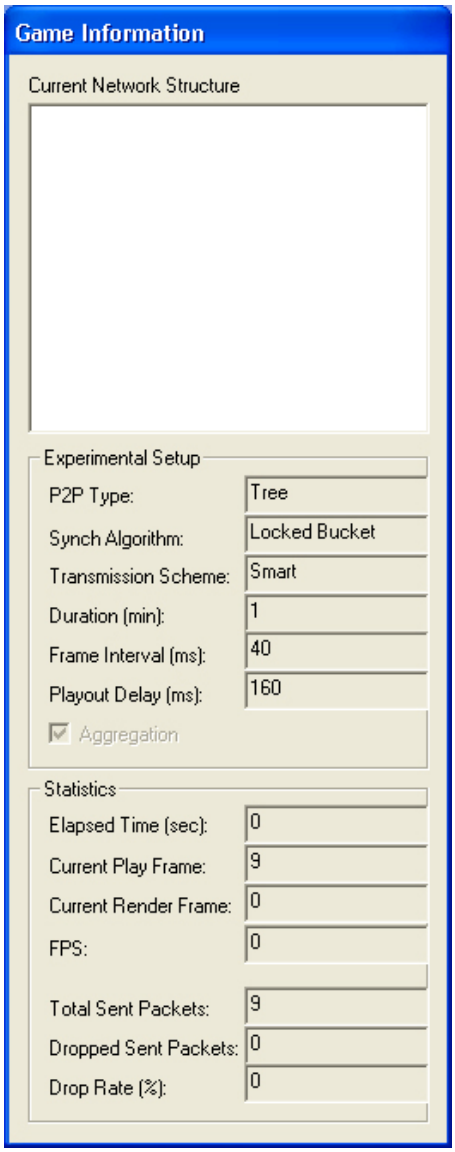


Figure 4.12: Game information dialog window of the game, Duel-x

For communication between players, each player's input is captured and packed into a packet and sent to other players during each frame. When other players send their packets, the packets will be received by this player and then the packets are stored into their game buffer and rendered together with their own frame data if no missing frame data are detected from all other players.

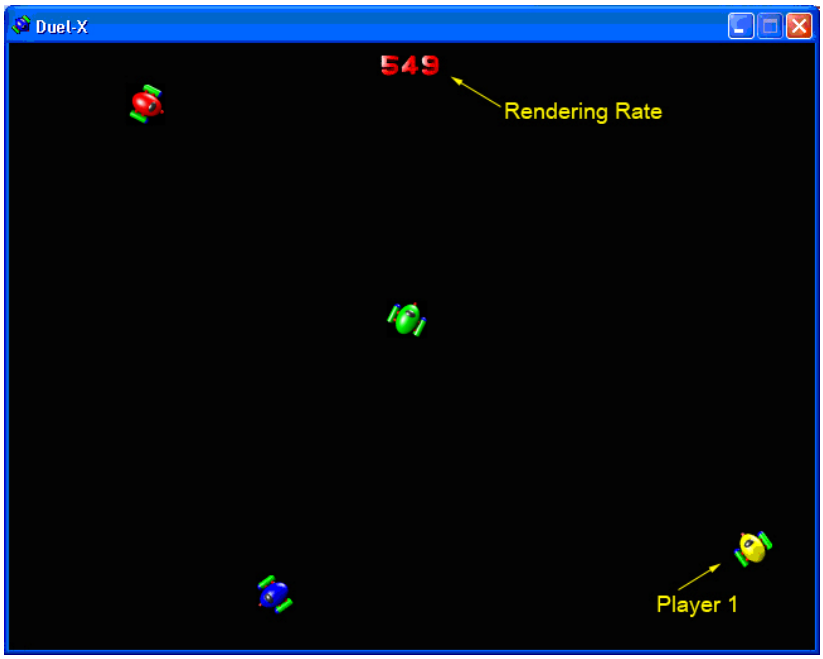


Figure 4.13: Game play screen of Duel-x

The game module is responsible of managing the states of its own player and others. In general, there are two types of data transfer: command and status. Sending status requires higher bandwidth requirement compared to transferring users' commands when the number of objects in the game increases. The Duel-X is capable of sending both data types to satisfy various experimental configurations. The following chapters will explain various approaches which are adopted in this game and show experimental results with tables and figures.

4.2.5. Networking Module

The network module transmits packets from its player and receives other players' packets through the Internet. TCP and UDP protocols are utilized for reliable and unreliable data transfer. There are mainly three tasks to perform for actualisation of this online game, namely, lobby related communication, RTT measurement, and game data transfer.

The first task can be divided into several sub-tasks, such as connecting to lobby system, creating a game room, retrieving game rooms list, joining the game, etc. These tasks are completed by using TCP for the easiness of implementation. Also, the task completion speed is not the major concern but the reliability is among these tasks. Therefore, the choice of using TCP can be appropriate in this case. However, measuring RTT requires using UDP because of the advantage of UDP such as low transfer overhead. Using TCP may delay the packet transferring due to the characteristics of TCP as explained in section 2.2.

Before actual game starts, several RTT measure packets are sent to target nodes and the replies from the target nodes are stored in the senders for the calculation of average RTT values to the nodes. These values are broadcasted to other nodes to create unified RTT table for a graph to tree conversion. The graph to tree conversion method will be explained in following chapters.

4.2.6. Experimental Data Set

It would be ideal situation for networking experiments if it is possible to provide commercial level of online games to users in all around world. However, due to resource constraints it is determined to utilise only eight nodes for experiments using Duel-X. Four of them are physically located in South Korea, one in Seoul and three in the Province of Jeju, a southern island in South Korea. The rest of them are sited in Griffith University, Australia. For the variety of network environment, the one node among four nodes in Griffith network accesses wireless network. Network information of these nodes is shown in **Table 4.4**.

Table 4.4: Node Information

Node	AU-1		**Node**	KR-1
IP	132.234.113.x		**IP**	61.39.243.x
Location	Gold Coast, Australia		**Location**	Jeju, Korea
CPU type	Intel Pentium 4, 1.60 GHz		**CPU type**	Intel Pentium 4, 2.8 GHz
Graphic Board	Radeon 9600 (VRAM: 256 MB)		**Graphic Board**	Radeon X1300 Pro (VRAM: 256 MB)
Memory	1 GB		**Memory**	1 GB
OS	Windows XP sp2		**OS**	Windows XP
Connection Type	T3		**Connection Type**	T3
Remarks	DELL GX-240 (lobby & web server)		**Remarks**	

Node	AU-2		**Node**	KR-2
IP	132.234.113.x		**IP**	61.99.159.x
Location	Gold Coast, Australia		**Location**	Jeju, Korea
CPU type	Intel Pentium 4, 2.0GHz		**CPU type**	Intel Celeron 1 GHz
Graphic Board	Intel 82845G (VRAM: 64 MB)		**Graphic Board**	Riva TNT (16 MB)
Memory	1 GB		**Memory**	256 MB
OS	Windows 2003 Server		**OS**	Windows XP
Connection Type	T3		**Connection Type**	Cable Modem (9 Mbps/1 Mbps)
Remarks	DELL GX-260		**Remarks**	

Node	AU-3		**Node**	KR-3
IP	132.234.113.x		**IP**	61.39.243.x
Location	Gold Coast, Australia		**Location**	Jeju, Korea
CPU type	Intel Pentium 4, 1.60 GHz		**CPU type**	Intel Pentium 4 1.70 GHz
Graphic Board	ATI Rage 128 pro (VRAM: 32 MB)		**Graphic Board**	GeForce FX5200 (VRAM: 128 MB)
Memory	512 MB		**Memory**	512 MB
OS	Windows XP sp2		**OS**	Windows XP sp2
Connection Type	T3		**Connection Type**	Cable Modem (9 Mbps/1 Mbps)
Remarks	DELL GX-240		**Remarks**	KCTV Cable

Node	AU-4		**Node**	KR-4
IP	132.234.113.x		**IP**	211.212.142.x
Location	Gold Coast, Australia		**Location**	Seoul, Korea
CPU type	Intel Pentium 4, 1.60 GHz		**CPU type**	AMD Athlon XP-A, 2200 MHz
Graphic Board	ATI Rage 128 pro (VRAM: 32 MB)		**Graphic Board**	NVIDIA GeForce2 MX/MX 400 (VRAM: 32 MB)
Memory	512 MB		**Memory**	512 MB
OS	Windows XP sp2		**OS**	Windows XP pro sp2
Connection Type	T3		**Connection Type**	Cable Modem (9.3 Mbps / 696 Kbps)
Remarks	DELL GX-240		**Remarks**	

4.3. Chapter Summary

This chapter describes two testing-bed applications that will be used for experiments with a number of approaches and algorithms proposed to ensure consistency across players, alleviate latency and reduce bandwidth requirements. The first application, COMP2P, is a network simulator which provides flexible and realistic functionality in a synthetic environment for wide-ranging explorations of graph and tree-based P2P systems, and packet transmission and aggregation schemes. It adopts Object Oriented Programming (OOP) concepts to create flexible and reusable modules (several modules are used without significant modification in the second testing-bed application, Duel-X).

COMP2P consists of three major classes, namely player, simulator, and statistics. The player class represents each participant in the game session. Each player object creates input, game, and re-sending buffers when game sessions start, to store packets from the other players, to fill up frame information for screen update, and to keep copies of packets transmitted to other players, respectively. Packet drop and delay are managed in the simulator class according to Poisson distribution and latency tables which store simulated transmission delay values. As its name implies, the statistics class gathers experimental data such as departure time, arrival time, resending frequencies, relay list, packet number, packet type, and game data for subsequent analysis.

To analyse the effects of proposed approaches and algorithms in a real-world environment, the study will also make use of a second testing-bed application, Duel-X, a multiplayer online game to be played in a dynamic environment such as the Internet. This application consists of four modules, namely game, networking, lobby, and web

server. Each player manages its own and the other players' status by using the game module. The game module transmits the data from the player by utilizing the networking module. The lobby module is responsible for providing a meeting place for players, managing a game room list, and providing the access information such as IP address and port number of the owner of each room. The web server provides services such as creating membership information, downloading applications, manuals, help, and so on. Finally, game data is collected and analysed by the MDX reader, an analysis tool developed especially for Duel-X.

By comparison with COMP2P, working with Duel-X involves greater complexity due to the need to manage system integrity in a dynamic environment. In particular, variables such as network latency, jitter, and bandwidth are hard to configure, predict and control in a real-world situation. Thus, while both testing-bed applications require significant investments of time and work for module development and debugging, Duel-X requires extra efforts to manage remote nodes.

Duel-X was originally developed as a 3D game based on Direct-X and Open-GL. At the time, the intention was to allow the game to be played freely by various people around the world. (For this reason, a lobby system and a web server are implemented to create accounts and distribute the game freely.) However, it soon became clear in experiments that several nodes with limited CPU and GPU processing powers suffered under the burden of 3D rendering, and ultimately the slow nodes degraded the overall performance of game sessions. Moreover, strengthened security policy on web access in Windows XP Service Pack Version 2 required major changes in the approaches to checking and launching of the game during the development process.

As a response, and in order to focus on the main issues pursued in this study, it was decided to change Duel-X to a 2D game, and to restrict participation at this stage to only eight nodes. These tactical decisions reduced CPU and GPU requirements and facilitated management of game parameter configuration and collection of experimental data. Strategically, it is intended that future research will return to the 3D game and will also allow the number of participants to be increased.

The source codes of COMP2, Duel-X modules, web server modules, and MDX reader are presented in Appendices A to D.

5. Consistency Maintenance Algorithms

An essential requirement of a multiplayer game system is the capacity to maintain consistency between the various players' states. In this chapter, a consistency maintenance algorithm, called Locked Bucket Synchronization (LBS), is proposed and compared with two conventional approaches, namely the Frequent State Regeneration (FSR) and the Lockstep (LS) algorithms.

FSR belongs to the group of 'optimistic' consistency maintenance algorithms, in the sense that as local and remote commands are received, they are immediately committed to processing. Any inconsistencies between commands (divergences between duplicated and real objects) are then resolved by means of data that are periodically transmitted from remote hosts. By contrast, both LS and LBS are in the 'conservative' group, in that they hold up game processing until the safety of operation commitments is assured. Thus, processing goes ahead only if the commands are consistent with one another. As can be expected, there is a trade-off between game execution speed on the one hand, and consistency across players on the other. The philosophy of LBS is to search for an appropriate balance between these two desirable, but competing, qualities.

The chapter is organised as follows. Section 5.1 presents a detailed description of the three consistency maintenance algorithms. Section 5.2 describes the experimental environment, and Section 5.3 reports the experimental results obtained from implementing the three algorithms. Section 5.4 compares the relative performances of these algorithms, and analyses in depth a number of related issues. Finally, Section 5.5 presents a summary of the main points raised in the chapter.

5.1. Alternative Consistency Maintenance Algorithms

In this section, the algorithm proposed to maintain consistency across players, Locked Bucket Synchronisation (LBS), is explained in detail. For subsequent comparisons of performance efficiency, two other consistency maintenance algorithms will be considered, namely Frequent State Regeneration (FSR), and Lockstep (LS). Both of the latter algorithms are also described in this section. Our discussion begins with the conceptually simplest algorithm, FSR.

5.1.1. Frequent State Regeneration Algorithm

The FSR algorithm is relatively straightforward to implement. According to this algorithm, each player's changes (made locally) are applied immediately to the objects. Those changes are then transmitted to the other players at regular intervals. Each player has two types of memory storage -- one for keeping other players' states and the other for the frame number of stored state data; see Figure 5.1. Whenever new packets arrive from other players, the frame number of the newly arrived packet is compared with the stored frame number. If the frame number of the packet is smaller than current frame number, the packet is out-ordered and its information is already superseded. In that case it is to be discarded. In Figure 5.1, the packets for frame 2 and frame 3 are out-ordered and the packet for frame 2 will be discarded. Information in packets that are not out-ordered is stored and used to update the screen with regard to the state of the owner of the packet.

This solution is perfectly suitable for certain network applications that concern about responsiveness rather than absolute synchronisation. Therefore, MODIGs, especially first person shooter (FPS), role-playing games (RPG), action/arcade games, are good

candidates for this solution in the network circumstance of high bandwidth and low latency. Due to one of main characteristic of FSR, regular state packet transmission, the more objects under control in the games, the higher bandwidth requirement for players is. Also, players may experience severe inconsistency that can not be easily resolvable caused by consecutive packet drops and long network latency.

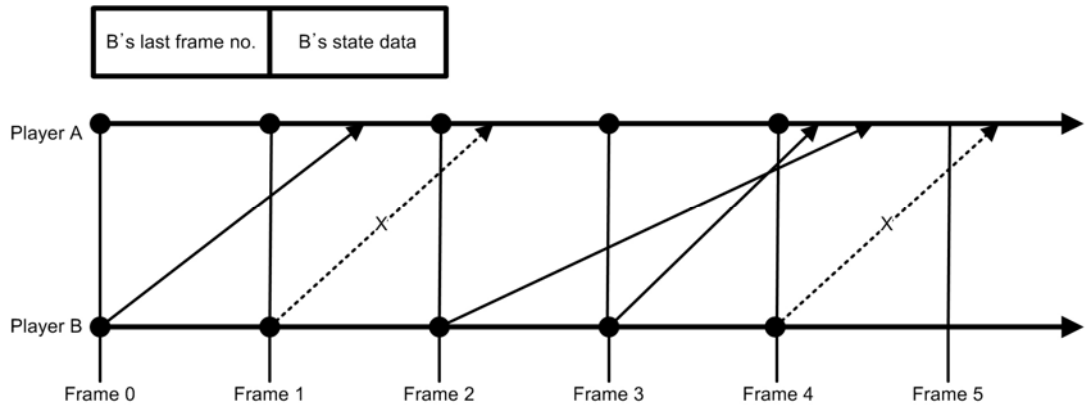


Figure 5.1: Frequent State Regeneration Algorithm

5.1.2. Lockstep Algorithm

The Lockstep algorithm (Baughman & Levine, 2001) is one of the simplest solutions for consistency maintenance in the P2P structure. Each peer waits for other peers' packets of current frame, makes its next move, sends packets and waits again. The drawback of this approach is that it can cause slowdown for game play if network latency is slower than the frame rate. For example, if the game's fps (frames per second) is 25 then each frame takes about 40ms to load. In case the network latency is longer than 40ms then players will have to wait until they get other players' packets.

Figure 5.2 shows the implemented version of lockstep algorithm in COMP2P network simulator and Duel-X multi-player online digital game. Each frame is associated with

a bucket which can hold the status or commands of game participants. States of local objects or local commands are also stored into frame buckets and the data in the buckets are used when current frame bucket is filled with all players' data. Rendering cycle and frame interval are separated to reduce playout delay in case of packet drops. Playout delay of frame 2 on player B is displayed in Figure 5.2. As it can be seen, when packet drop happens it severely delay the game execution, especially the network latency between two nodes is relatively longer than others. To reduce the frequency of packet transferring, input gathering and packet transmission are coupled with the frame interval which is 40 ms in the two experimental testing beds, COMP2P and Duel-X.

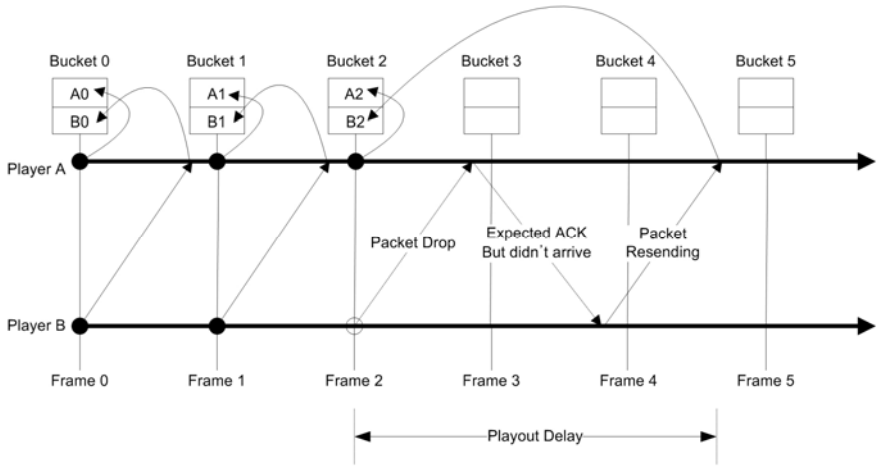


Figure 5.2: Lockstep Algorithm

5.1.3. Locked Bucket Synchronisation Algorithm

One of the main problems of the Lockstep algorithm is the irregular playout delays due to network latency. The playout delay is the time difference between when players' commands are generated and when they are executed and appeared on the players' screen. Responsiveness or response time is an alternative term for playout

delay (Mauve et al., 2004). To prevent irregular playout delays, the Bucket-Synchronization algorithm extends the playout delay. It is analogous to the buffering mechanism of streaming audio and video. As displayed in Figure 5.3, commands issued between Frame 0 and Frame 1 are stored in Bucket 0 and executed at Frame 2. The major difference between the bucket-synchronization algorithm (BSA) and the locked bucket-synchronization algorithm (LBSA) is the mechanism to handle inconsistency when it happens. In that case, the former simply ignores it or convergence process begins when the threshold of state difference between players is exceeded. The latter adopts the method of the Lockstep algorithm which is send-and-wait mechanism. In the LBSA, the process waits until current frame's corresponding bucket is filled with packets of all players. When the delayed packet arrives, it is stored in the corresponding bucket and the player's game process moves forward again. To prevent a dead-lock situation (Wolfson, 1987), each player sends packets at every frame even if there is no command to transmit. In Figure 5.3, the empty circle represents packets without commands.

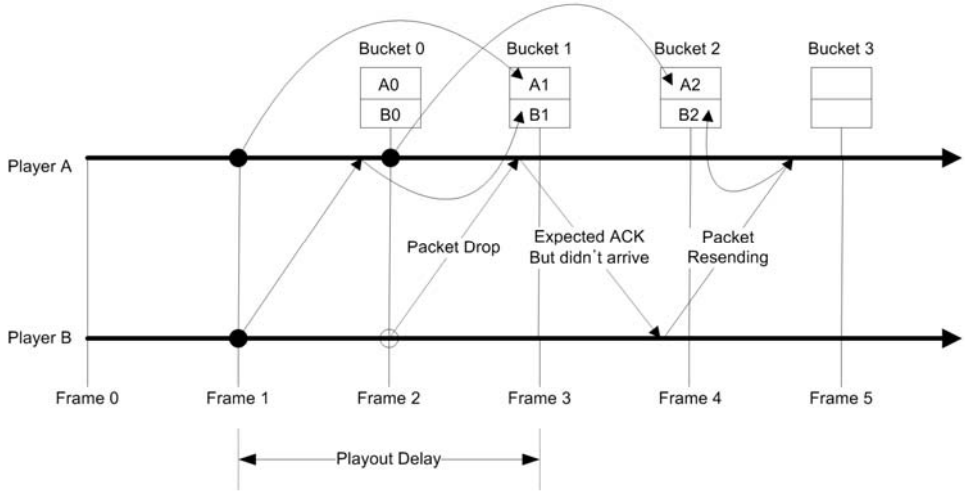


Figure 5.3: Locked Bucket-Synchronisation Algorithm

Similar algorithm is suggested in Baughman & Levine (2001). However, the algorithm is used for network game security and introduces additional network latency or processor cycles. The LBSA commits its commands in corresponding buckets without using the two-phase commitment described in Baughman & Levine (2001). Because it assumes the lookahead cheat can be detected using RTT (Round-trip time) measurement methods. For example, It is assumed that player A's IP is 132.234.xxx.5 and player B, 61.41.yyy.3. Each player measures RTT to the opponent's IP address and neighboring IP addresses before game session starts. When the opponent reports unusual network latency then the player may be performing the lookahead cheat. Then by agreement between other players, the cheat player will be banned from the game session.

5.2. Experimental Environment

The FSR, LS and LBS algorithms are evaluated and compared to each other by using COMP2P and Duel-X in this section. A detailed description of the experimental environment is presented in this section.

An Intel Pentium 4 equipped machine, with 1.60GHz CPU and 512MB RAM, under Microsoft windows XP environment was used for the experiments on COMP2P network simulator. For the experiments on Duel-X, maximum eight computers are used. Four of them are located in South Korea and others are located in Griffith University, Australia. For detailed information about the hardware and software specifications for the computers used in the experiments, please refer to Section 4.2.6.

Three groups of experimental data set, comprising two players, four players, and eight players, are organised to compare the effect of the number of players in game sessions

by various game playability measure factors, such as game execution speed, bandwidth requirement, network jitter, network latency, and so on. Table 5.1 shows the group id, nodes, and justification of each group in detail. The game execution speed is measured by frames per second (FPS). Bandwidth requirement is gauged by packet numbers per second (PPS) rather than bits per second (bps) for the simplicity of calculation. The PPS will be divided into two groups - inbound and outbound PPS. As the names implies, inbound PPS refers to the number of packets which are transferred from other nodes and outbound PPS refers to the number of packets which are sent to other nodes. Even the size of packets depends on the types of packets, such as ACK for acknowledgement, and Game for command and state packets.

Table 5.1: Node Groups for Experiments

4 players

Group ID	Nodes	Remarks
G4-1	AU-1, AU-2, AU-3, AU-4	Experiments on LAN
G4-2	KR-1, KR-2, KR-3, KR-4	Domestic WAN
G4-3	AU-1, AU-2, KR-1, KR-2	International WAN

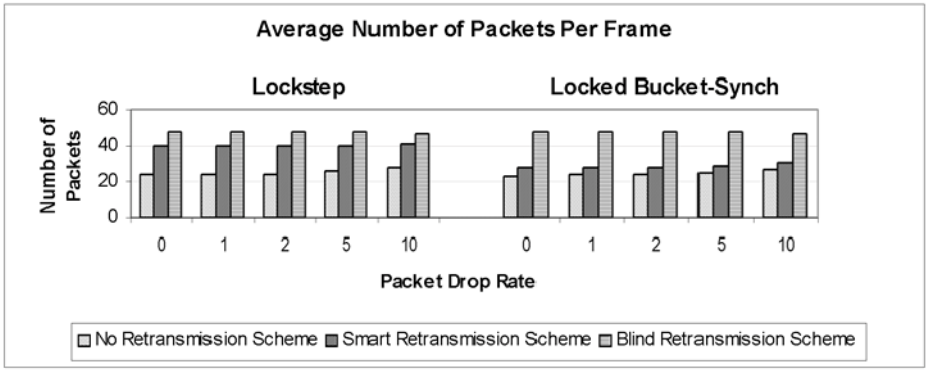
8 players

Group ID	Nodes	Remarks
G8-1	AU-1 ~ 4, KR-1 ~ 4	International WAN

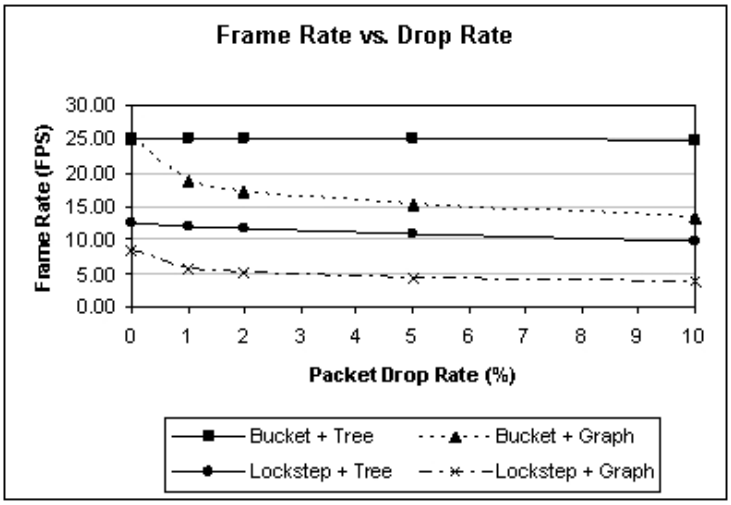
5.3. Experimental Results

Some of the results reported below have been presented in a work-in-progress format at several conferences (Moon et al., 2005; Moon et al, 2006a; Moon et al., 2006b; Moon et al., 2006c). Figure 5.4 (a) shows the changes of average number of packets per frame according to packet drop rate on LS and LBS algorithms with the proposed transmission schemes. The LS and LBS algorithms are utilised for the efficiency comparisons between graph and tree-based P2P systems in Figure 5.4 (b). These

results are obtained from the experiments on the COMP2 network simulator with the experimental data set explained in Section 4.1.5.



(a) Experimental results from Transmission Schemes



(b) Experimental results from Graph and Tree-based P2P systems (16 players)

Figure 5.4: Experimental results from proposed approaches

5.3.1. Experimental results on COMP2P with FSR, LS and LBS

From the above figure, each graph contains experimental results of LS and LBS algorithms on each approach. Therefore, the experimental results on solely for LS and LBS will be presented in the following sections. The experimental results on FSR also excluded because the characteristics of FSR determines the neglect of absolute

consistency and network latency. Simply FSR neglects packet drops and network latency, so it achieves upper benchmark frame speed, in this case 25 FPS, without considering packet drops and network latency. Also, the COMP2P network simulator is not equipped with inconsistency analysis functionality in game data processing. Consequently, another experimental platform - Duel-X, is used to analyse the efficiency of three consistency maintenance algorithms, FSR, LS and LBS.

As it is mentioned previously, total four groups of nodes are used for the experiments and the performance comparisons which will be presented in this section. To provide an overview of each experiment, FPS (frame per second), PPS (packets per second), packet drop rate, packet resending rate are displayed for each group and algorithm. Then, specific cases are selected to explain the effect of several key parameters on the game execution efficiency.

5.3.2. Experimental results on Duel-X with FSR

Group G4-1: (Pre-measured One-Trip Time)

Before starting game session, OTT (One-Trip Time) is measured between nodes for two purposes. The fist one is for general analysis such as relationship between OTT and frame interval, OTT and FPS, and so on. Also, sometimes using this application, Duel-X, is the only solution to measure OTT time between nodes because most of educational organisations disable ICMP packets, especially PING purposes in their firewall setting. The second purpose is that OTT values are utilised for packet resending mechanism in LS and LBS algorithms. The OTT values between nodes in group G4-1 are listed in Table 5.2 in which the maximum OTT value between node AU2 and node AU4 is 10.09 ms and it will not affect much on game playability

because pre-defined frame interval is 40 ms. The relationship between OTT and FPS will be examined in the following sections.

Table 5.2: One-Time Time between nodes in G4-1

Node No.	AU1	AU2	AU3	AU4
AU1		8.07	4.83	6.54
AU2	8.07		7.17	10.09
AU3	4.83	7.17		3.85
AU4	6.54	10.09	3.85	

Group G4-1: (Overall Performance)

First, the experimental results from LAN environment, from group G4-1, are presented. Table 5.3 shows experimental parameters, which utilises FSR as a synchronisation algorithm, four total players, and 40ms frame interval.

Table 5.3: Parameters for Group G4-1

P2P Type:	Mesh
Synch Algorithm:	FSR
Transmission:	None
Aggregation:	no
Frame Interval (ms):	40
Playout Delay (ms):	
Total Players:	4

FPS (frame per second), PPS (Packets per second), and drop rate values are grouped in Table 5.4 from the experiments of group G4-1 with the use of FSR algorithm. These results are averaged from the experiments which are repeated ten times. As mentioned in previous sections, FSR does not send acknowledgement, therefore, drop rates are measured by exclusively use of incoming packets. The word "IN" denotes that the values are related to inbound packets.

FPS values show that how many frames are executed per second in each node and it is directly related to PPS (Packets per Second) values, especially to PPS (OUT) which is the number of packets that sent to other nodes per second. Because FRS algorithm

transfers packets without concerning other nodes performance, the FPS values show the upper benchmark performance of each node. These values will be used for the comparisons between other algorithms and diversion analysis with frame interval values later on.

Table 5.4: FPS, PPS, Drop Rate in G4-1 with FSR

Node	FPS:	Drop Rate (IN)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	24.89	0.03%	63.94	74.73	138.67
AU2	21.33	0.02%	67.48	64.03	131.51
AU3	21.26	0.04%	67.58	63.83	131.40
AU4	21.32	0.02%	67.51	64.02	131.53

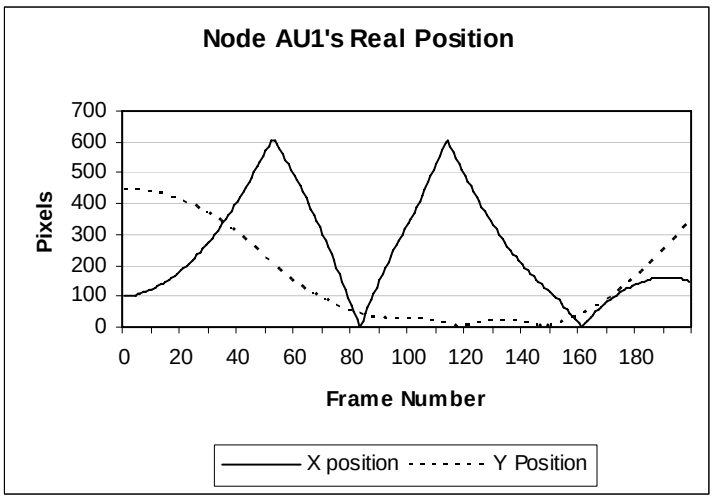
Group G4-1: (Divergence)

To provide clear analysis for divergence of real and ghost objects between nodes, auto-pilot function is implemented. The real objects denote that the objects which reside in the owner of the objects. The ghost objects represent duplicated objects in other players that are copied from real objects in the owner player of the objects. The screen size is 640 pixels of width and 480 pixels of height. Ships can not travel beyond the screen border and they bound back when they reach the boundary of the screen.

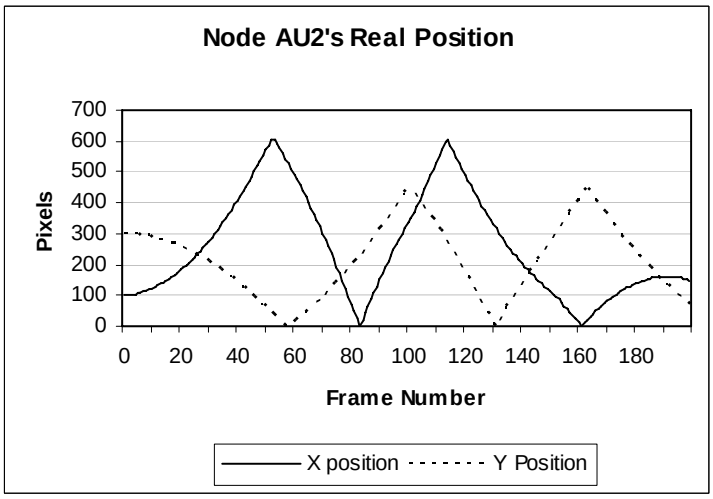
From Figure 5.5 (a) to (d) displays the real positions of ships in node AU1 to AU4 respectively. The position of real object in each node is transferred to other nodes on regular basis and the frequency is recorded as FPS. Pre-defined frame interval value controls the frequency, however, it is heavily related to the system performance of each node. The system performance is directly related to hardware and software that resides in each node such as the type of video cards, the amount of memory, windows

configuration, and so on. Also, network environment such as traffic congestion, network hardware and so on can be effective on this situation.

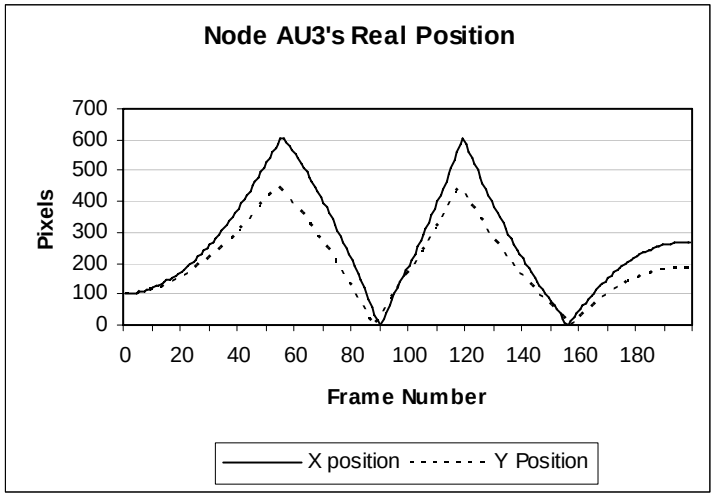
Two cases are presented here to show the factors that affect the divergence of real and ghost object on game play in LAN environment. The first factor is network latency, which does not have significant effect in this case. The result of the experiments that occur in LAN environment and RTT (Round-Trip Time) between nodes is less than 10 ms according to the experiments that are presented in the following sections.



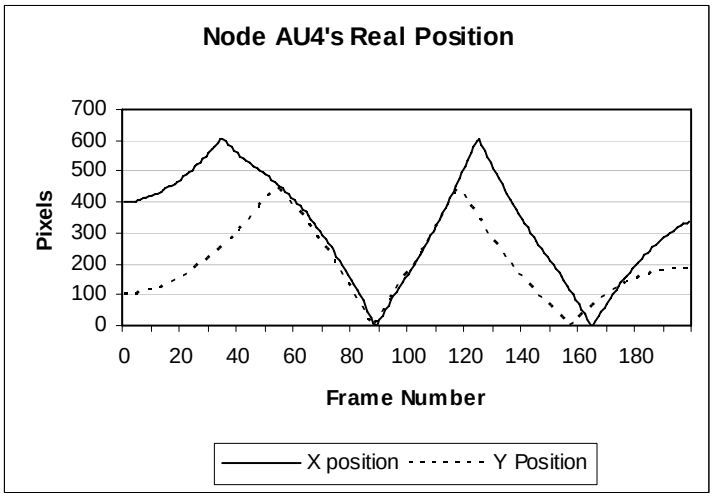
(a) AU1's ship position (real object)



(b) AU2's ship position (real object)



(c) AU3's ship position (real object)



(d) AU4's ship position (real object)

Figure 5.5: The positions of real objects

Table 5.5 and Figure 5.6 show the difference of positions of real object in node AU2 and the position of ghost object in node AU4, respectively. The divergence of real object and ghost object is negligible at 0.67 pixels for x position and 0.49 for y position. FPS values for both nodes are the same that is 21.33. The case of same or similar FPS values with long latency will be presented in following sections.

Table 5.5: AU2's divergence of real and ghost object in node AU4

	Divergence X	Divergence Y
AVG:	0.67	0.49
STD:	2.96	2.13
Median	0.00	0.00
MAX	23	17
MIN	0	0

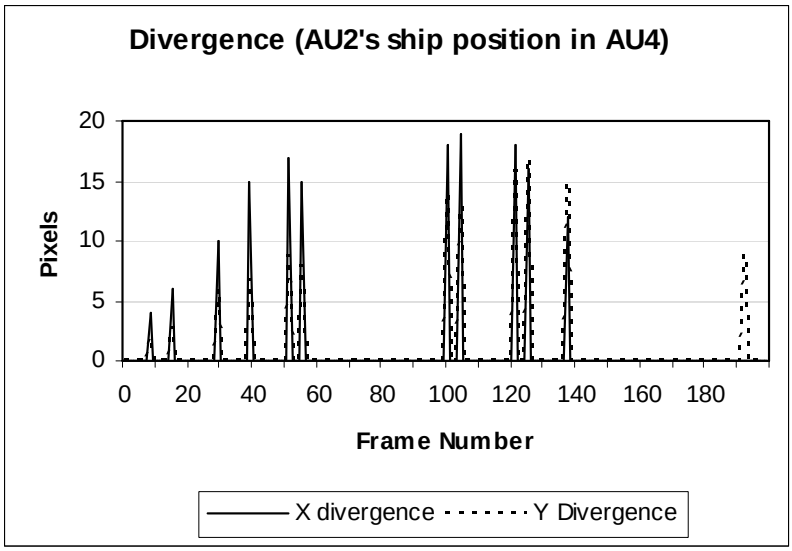


Figure 5.6: The position difference between AU2's real object and ghost object in node AU4

The second factor is FPS and it has significant effect on the divergence between real and ghost objects. Table 5.6 and Figure 5.7 show the position difference between node AU1's ship in node AU1 (real object) and node AU1's ship in node AU4 (ghost object). Even though these two nodes reside in the same LAN and RTT values for both nodes are less than 10 ms, the average divergence of position x is 175.80 pixels and that for position y is 136.85 pixels. As can be seen in Figure 5.7, the divergences of position x and y fluctuate due to the bouncing movement of the ships. The FPS value for node AU1 is 24.93 and the FPS value for node AU4 is 21.33 in this case. This implies that not only is reducing or masking network latency important to reduce divergence of real and ghost objects but also FPS or game execution speed is. The

third factor which is game starting time synchronisation will be covered in the following sub-sections.

Table 5.6: AU1's divergence of real and ghost object in node AU4

	Divergence X	Divergence Y
AVG:	175.80	136.85
STD:	124.28	103.43
Median	158.00	121.00
MAX	607	444
MIN	0	0

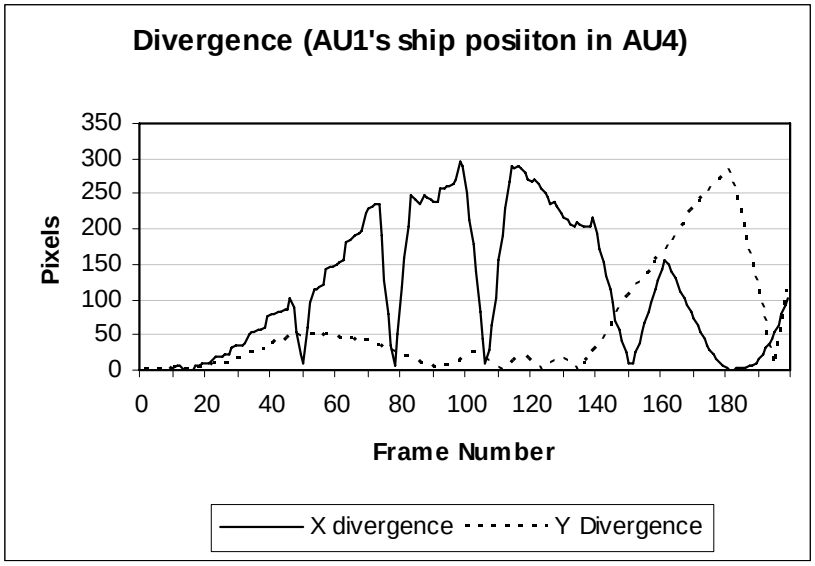


Figure 5.7: The position difference between AU1's real object and ghost object in node AU4

Table 5.7 shows the average divergence values of real and ghost objects in each node for Group G4-1. As it can be seen in the table, the AU2-AU4 pair has the lowest divergence values of 1.01 and 11.97 among AU pairs. The difference between divergence values of AU2-AU4 and AU4-AU2 pairs is from unsynchronised game starting time. It will be examined in the following sections.

Table 5.7: Divergence of real and ghost object in Group G4-1

Real Obj.	Ghost Obj.	Div. X (Avg.)	Div. Y (Avg.)	Div. X (STD)	Div. Y (STD)
AU1	AU2	175.57	137.11	0.80	0.60
AU1	AU3	178.20	139.87	0.33	1.20
AU1	AU4	175.72	137.22	0.84	0.63
AU2	AU1	174.90	136.66	0.73	0.46
AU2	AU3	39.20	28.66	2.61	1.66
AU2	AU4	1.01	0.76	0.53	0.42
AU3	AU1	186.36	136.14	0.61	0.50
AU3	AU2	50.53	37.05	2.92	2.19
AU3	AU4	49.68	36.42	2.87	2.12
AU4	AU1	184.83	136.82	0.49	0.25
AU4	AU2	11.97	8.88	0.03	0.02
AU4	AU3	35.67	26.34	0.94	0.95

Group G4-1: (Frame Interval)

When game packets are sent to other nodes, these packets are stored with time information such as packet arrival time in the other nodes as a form of binary file to save space and for further analysis. Frame interval is one of analysed items and it shows the regularity of packet arrival for each node.

Table 5.8 shows the average, standard deviation, and median of frame interval for each node. As indicated in the table, average frame interval values for each node range from 40.10 ms to 46.99 ms, similarly to the pre-defined frame interval of 40 ms.

Figure 5.8 shows the frame interval of node AU1 measured in node AU2. The average frame interval in this instance is 40.11 ms with a standard deviation of 3.73 ms which is very close to the average values in Table 5.8. The figure also shows that the regularity of packet arrival interval is high indicating that the game playability is very good in terms of the smoothness of ghost object's movement.

More radical changes in frame interval can be observed in Figure 5.9. The averaged frame interval is 46.99 ms and standard deviation of that is 16.78 ms. The figure

shows that it has more and higher pikes than Figure 5.8 and this implies that the playability of this game in node AU4 may be affected by the smoothness of the movement of node AU3's ship due to the irregularity of packet arrival.

Table 5.8: Frame interval in Group G4-1

Destination	Origin	FI AVG	FI STD	FI Median
AU1	AU2	46.88	5.49	48.00
AU1	AU3	46.99	12.00	47.00
AU1	AU4	46.88	3.41	46.00
AU2	AU1	40.10	4.05	40.00
AU2	AU3	46.99	13.74	47.00
AU2	AU4	46.89	4.49	47.00
AU3	AU1	40.11	11.90	46.00
AU3	AU2	46.88	14.21	47.00
AU3	AU4	46.88	9.12	47.00
AU4	AU1	40.10	8.33	46.00
AU4	AU2	46.89	11.87	47.00
AU4	AU3	46.99	15.39	47.00

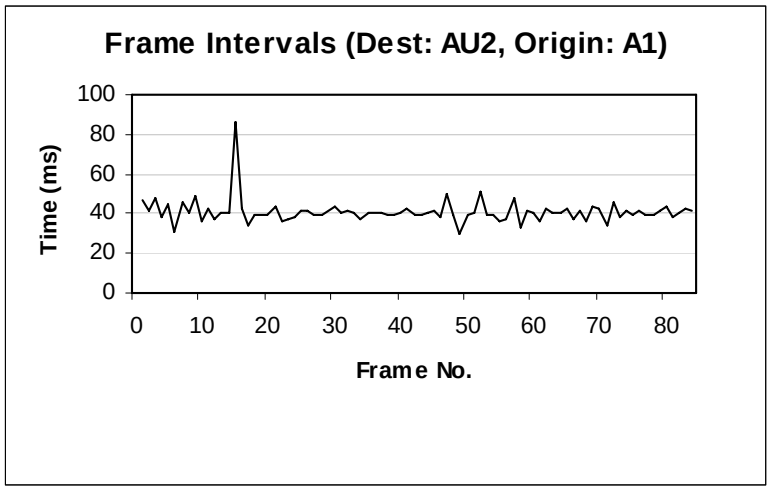


Figure 5.8: Frame Interval of node AU1 in node AU2

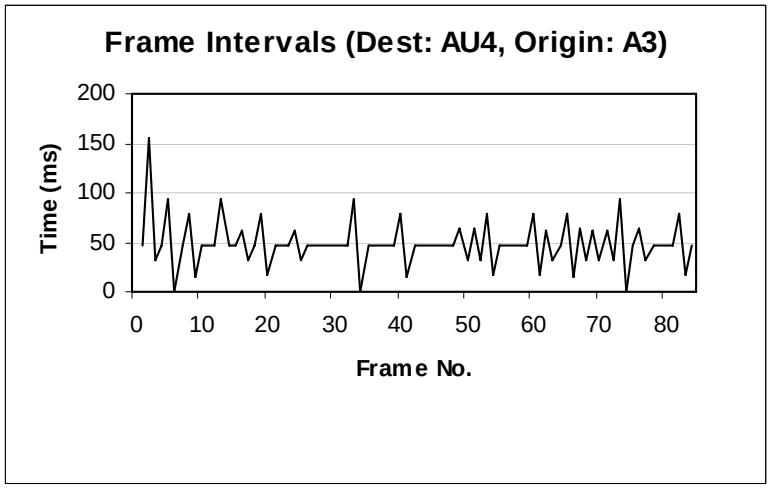


Figure 5.9: Frame Interval of node AU3 in node AU4

Group G4-2: (Pre-measured One-Trip Time)

OTT (One-Trip Time) values between nodes in group G4-2 are presented in Table 5.9. As denoted in the table, the highest OTT value is from node KR2 which is 42.93 ms and it is slightly higher than pre-defined frame interval value. The effect of this will be examined in the following sub-sections. The lowest OTT value of 10.04 ms is from KR1 and KR3 pair.

Table 5.9: One-Time Time between nodes in G4-2

Node No.	KR1	KR2	KR3	KR4
KR1		40.15	10.04	25.36
KR2	40.15		41.65	42.93
KR3	10.04	41.65		28.27
KR4	25.36	42.93	28.27	

Group G4-2: (Overall Performance)

This section presents the experimental results from second group G4-2. G4-2 consists of four nodes that reside in South Korea. Three of them are located in geographically close areas (the Province of Jeju: a southern island in South Korea), and the other is in

Seoul (the capital city of South Korea). Parameters for the experiments are same as those of group G4-1 as shown in Table 5.3

Information of the overall system performance, packet drop rate, and packet transfer rate is presented in Table 5.10. The table denotes that node KR1 and KR3 have similar FPS values which are 21.33 and 21.31. The packet drop rates are similar to the results from group G4-1 ranging from 0% to 0.04%. Packets per second (PPS) values are directly related to both FPS values and number of nodes to transfer packets. Calculation methods for PPS (IN) and PPS (OUT) will be provided in the analysis section.

Table 5.10: FPS, PPS, Drop Rate in G4-2 with FSR

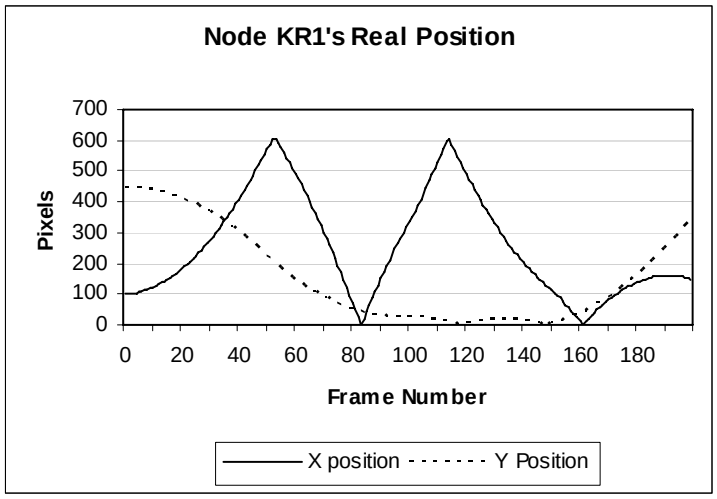
Node	FPS:	Drop Rate (IN)	PPS (IN)	PPS (OUT)	PPS (ALL)
KR1	21.33	0.03%	50.86	64.05	114.90
KR2	15.12	0.00%	56.61	45.42	102.03
KR3	21.31	0.02%	50.88	63.99	114.87
KR4	14.32	0.04%	57.85	43.00	100.85

Group G4-2: (Divergence)

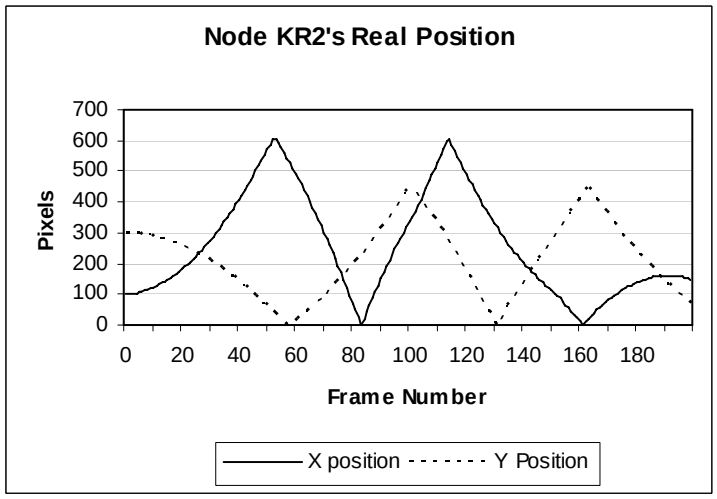
The real objects' positions for Node KR1 to KR4 are presented in Figure 5.10 (a) to (d), respectively. These objects also show bouncy movement as demonstrated in the figures.

To show the effect of unsynchronised game execution speed, the case of divergence measurement between node KR1 and node KR2 is presented here. The FPS value of node KR1 is 21.33 and node KR2's one is 15.12 (see Table 5.10). Therefore, KR1's ship information is transferred to node KR2 more frequently than KR2's. Figure 5.11 (a) shows KR1's real object position and Figure 5.11 (b) displays ghost object position. The divergence between real and ghost objects in node KR1 is shown in

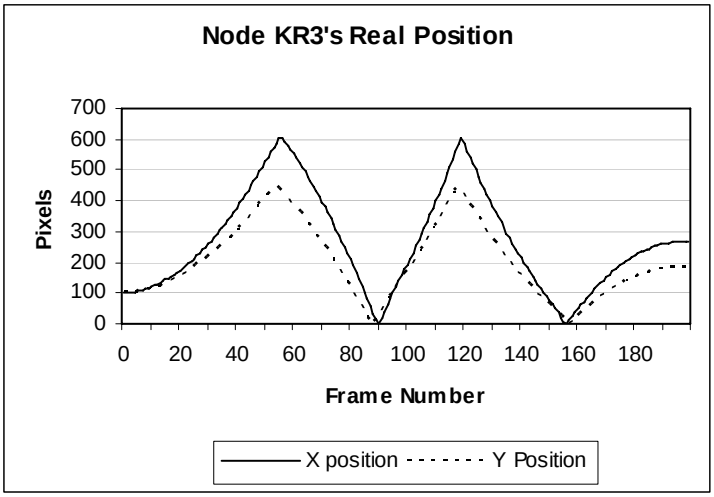
Figure 5.11 (c). Divergence of KR1's real object from its ghost object in node KR2 is presented in Table 5.11.



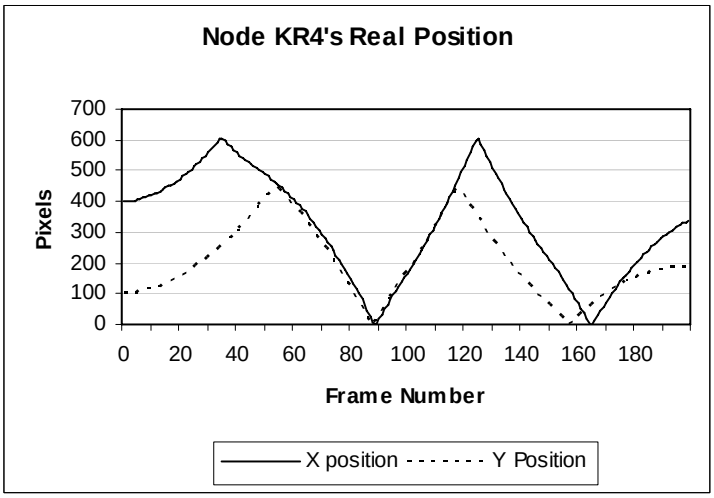
(a) KR1's ship position (real object)



(b) KR2's ship position (real object)



(c) KR3's ship position (real object)



(d) KR4's ship position (real object)

Figure 5.10: The positions of real objects

Table 5.11: KR1's divergence of real and ghost object in node KR2

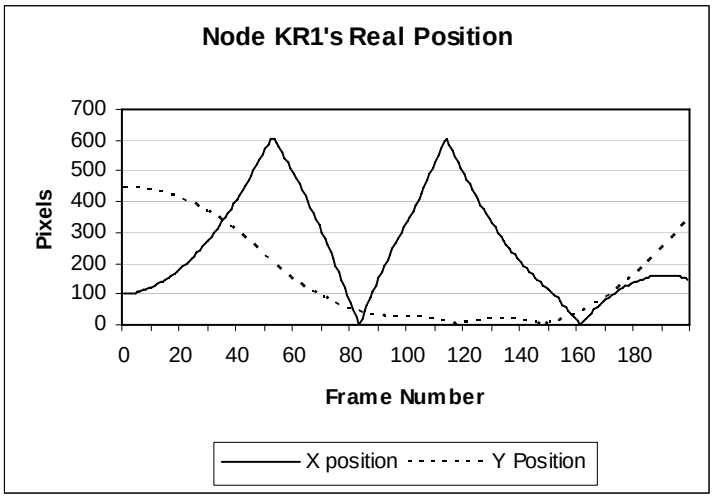
	Divergence X	Divergence Y
AVG:	180.75	121.44
STD:	133.24	98.00
Median	163.00	93.00
MAX	607	447
MIN	0	0

Figure 5.10 (b) is a compressed form of Figure 5.10 (a) because node KR1 sends packets more frequently than node KR2's frame speed. The shape of the line is almost

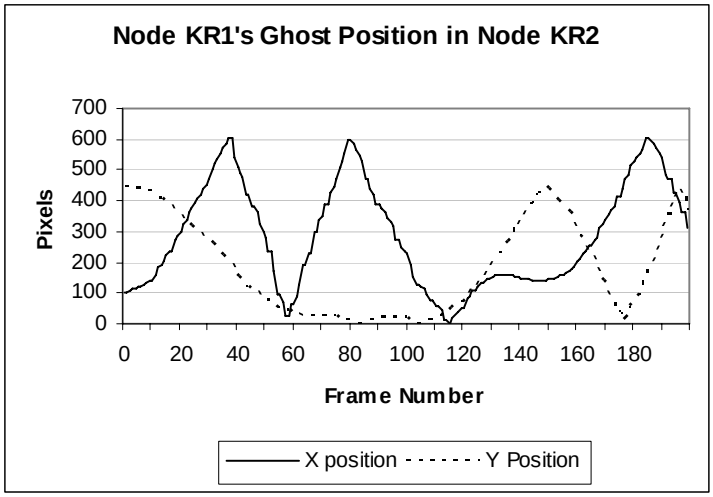
identical, however, ghost object position lines are rougher than real object position graph's. This is caused by the simple mechanism of using previous ship location when packets are delayed or dropped. To compare this situation with opposite situation, Figure 5.12 is presented and it shows the case of node KR2 and KR1 pair. As mentioned previously, node KR2's FPS value is smaller than KR1's. Therefore, KR2's packets are sent to KR1 less frequently than the frame speed of KR1. It causes more frequent use of previous packets of node KR2 in node KR1. As a result, the graph of ghost object position of node KR2 in node KR1 is rougher than the graph of ghost object position of node KR1 in node KR2 as is shown in Figure 5.12 (b). Divergence of KR2's real object from its ghost object in node KR1 is presented in Table 5.12.

Table 5.12: KR2's divergence of real and ghost object in node KR1

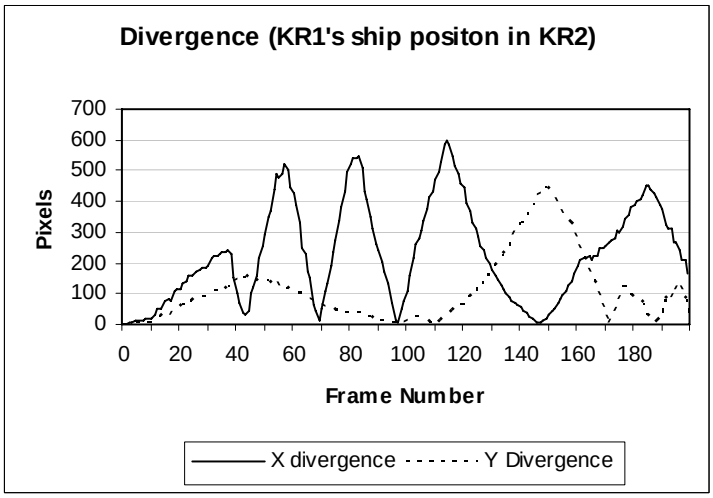
	Divergence X	Divergence Y
AVG:	194.38	151.05
STD:	132.98	100.21
Median	177.00	134.00
MAX	597	447
MIN	1	0



(a) Node KR1's real object position



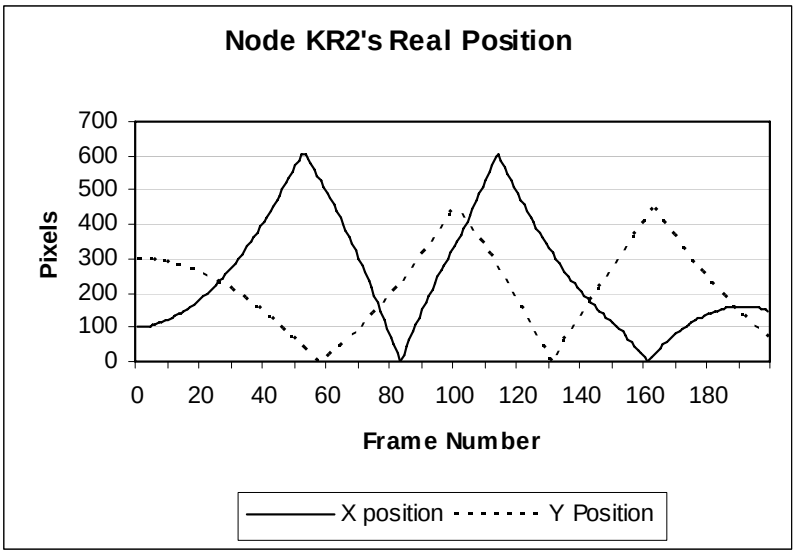
(b) Node KR1's ghost object position in node KR2



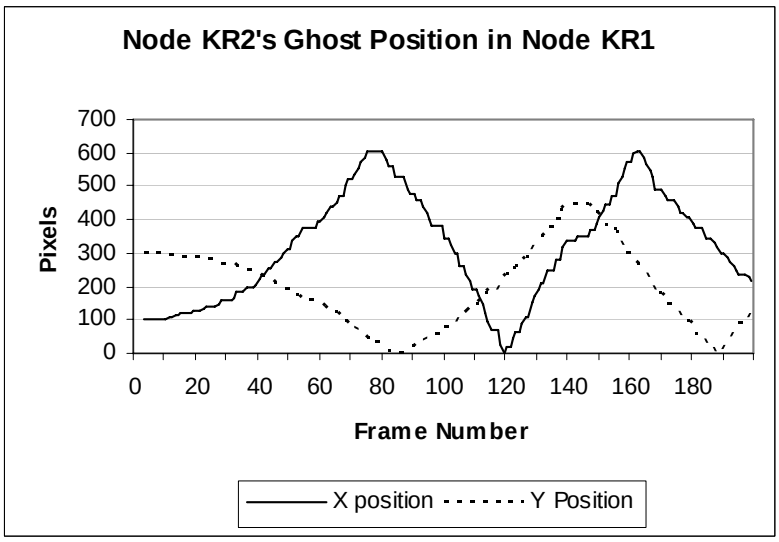
(c) Position difference between real object of KR1 and ghost object in KR2

Figure 5.11: Divergence of KR1's real and ghost objects

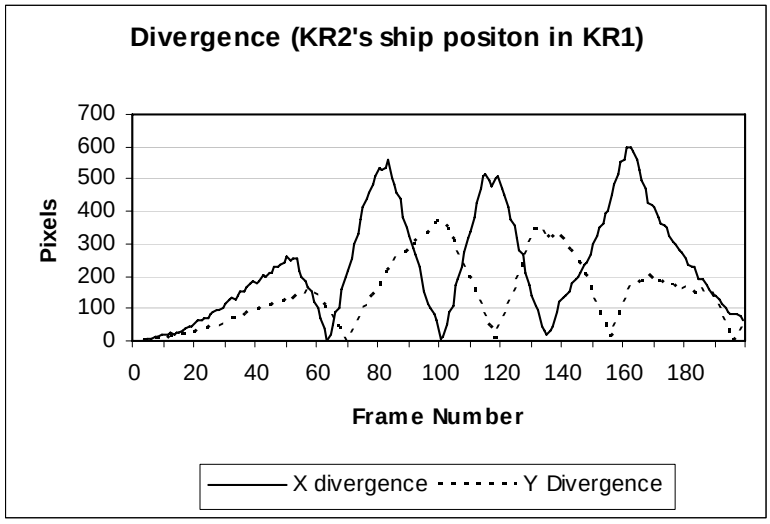
The average divergence values of real and ghost object in each node for group G4-2 are presented in Table 5.13. As it can be seen in the table, the KR1-KR3 pair has the lowest divergence values among other pairs which are 6.21 and 18.27 pixels. Again, the difference of divergence values between KR1-KR3 and KR3-KR1 pairs are from unsynchronised game starting time.



(a) Node KR2's real object position



(b) Node KR2's ghost object position in node KR1



(c) Position difference between real object of KR2 and ghost object in KR1

Figure 5.12: Divergence of KR2's real and ghost objects

Table 5.13: Divergence of real and ghost object in Group G4-2

Real Obj.	Ghost Obj.	Div. X (Avg.)	Div. Y (Avg.)	Div. X (STD)	Div. Y (STD)
KR1	KR2	181.44	134.23	4.76	1.43
KR1	KR3	6.21	4.29	2.49	1.89
KR1	KR4	179.09	142.07	1.27	1.19
KR2	KR1	188.88	154.65	4.23	7.52
KR2	KR3	188.59	154.95	4.79	7.24
KR2	KR4	154.01	122.70	7.53	8.34
KR3	KR1	18.27	13.40	4.05	3.06
KR3	KR2	179.79	133.29	6.66	4.42
KR3	KR4	190.05	140.48	5.85	4.20
KR4	KR1	183.09	142.64	9.94	1.79
KR4	KR2	168.64	136.54	6.72	4.23
KR4	KR3	181.95	141.27	6.31	1.81

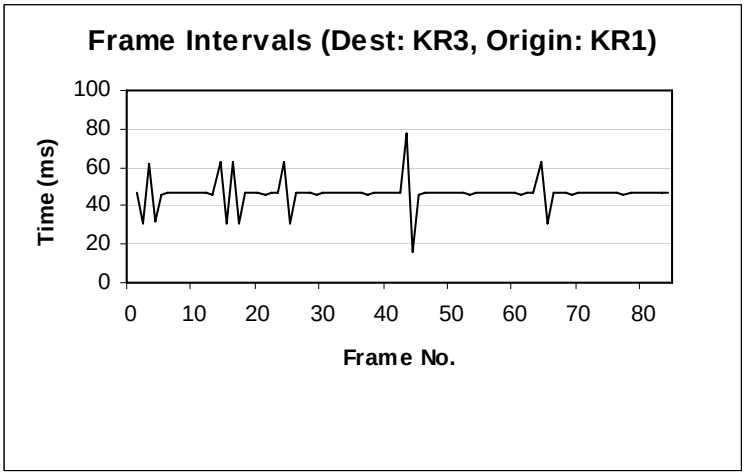
Group G4-2: (Frame Interval)

The average, standard deviation, and median values of frame interval for each node in group G4-2 are shown in Table 5.14. Shown in the table, average frame interval values for each node range from 46.74 ms to 73.60 ms. The minimum frame interval value is close to the pre-defined frame interval, 40 ms. The maximum value is generated from node KR4, and other frame interval values measured in node KR1,

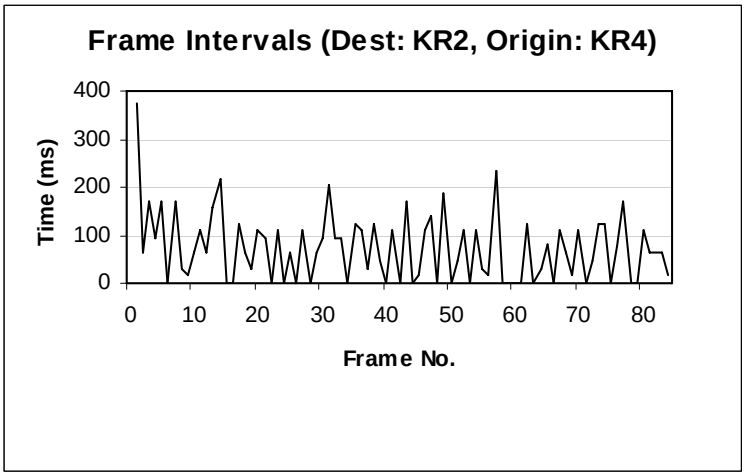
KR2, and KR3 for the packets from KR4 are close to the maximum value. This indicates that the node KR4 performs poorly in terms of frame execution speed as shown in Table 5.14. (FPS, PPS, Drop Rate in G4-2 with FSR). In terms of regularity of frame execution and packet transfer, node KR3 and KR1 pair shows the best performance among other pairs of nodes measured by the standard deviation values shown in Table 5.14 and Figure 5.13 (a). On the other hand, the node pair, KR2-KR4, shows the worst performance in terms of game execution speed and the regularity of it presented in Figure 5.13 (b).

Table 5.14: Frame interval in Group G4-2

Destination	Origin	FI AVG	FI STD	FI Median
KR1	KR2	65.56	53.50	47.00
KR1	KR3	46.89	10.34	47.00
KR1	KR4	73.27	46.07	78.00
KR2	KR1	47.10	54.24	31.00
KR2	KR3	47.11	54.75	31.00
KR2	KR4	73.60	71.99	70.00
KR3	KR1	46.84	8.01	47.00
KR3	KR2	65.52	53.05	47.00
KR3	KR4	73.28	44.62	78.00
KR4	KR1	46.89	51.98	16.00
KR4	KR2	65.39	70.03	47.00
KR4	KR3	46.74	51.98	16.00



(a) Frame Interval of node KR1 in node KR3



(b) Frame Interval of node KR4 in node KR2

Figure 5.13: Frame Interval of node KR1 and KR3

Group G4-3: (Pre-measured One-Trip Time)

Group G4-3 consists of four nodes, node AU1, AU2, KR1, and KR3, two of which exist in Australia and the other in South Korea. All of them are to analyse the performance of game execution speed and packet transfer in international WAN environment. The OTT (One-Trip Time) values between nodes in group G4-3 are measured and presented in Table 5.15. It takes more than 160 ms to transfer packets from Australia to South Korea and less than 10 ms between nodes in LAN or nearly LAN environment.

Table 5.15: One-Time Time between nodes in G4-3

Node No.	AU1	AU2	KR1	KR3
AU1		7.67	164.86	176.38
AU2	7.67		168.86	166.54
KR1	164.86	168.86		13.33
KR3	176.38	166.54	13.33	

Group G4-3: (Overall Performance)

The overall system performance, packet drop rate, and packet transfer rate are presented in Table 5.16. Node AU1 shows almost upper benchmark FPS value which is 24.90 and other three nodes perform nearly identical performance in terms of game execution speed which is about 21.32 frames per second. The packet drop rates are similar to the results from group G4-1 and G4-2 which ranges 0% to 0.04%. FPS values are well reflected on PPS (IN) and PPS (OUT) values as shown in the above table. For example, node AU1's PPS (OUT) value is 74.76. This value can be easily calculated from its FPS value of 24.9, multiplied by the number of nodes that the node AU1 transfers its packet to. More detail will be covered in the following analysis sections.

Table 5.16:FPS, PPS, Drop Rate in G4-3 with FSR

Node	FPS:	Drop Rate (IN)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	24.90	0.04%	64.04	74.76	138.80
AU2	21.32	0.00%	67.55	64.02	131.57
KR1	21.33	0.02%	67.30	64.05	131.35
KR3	21.31	0.02%	67.33	63.98	131.31

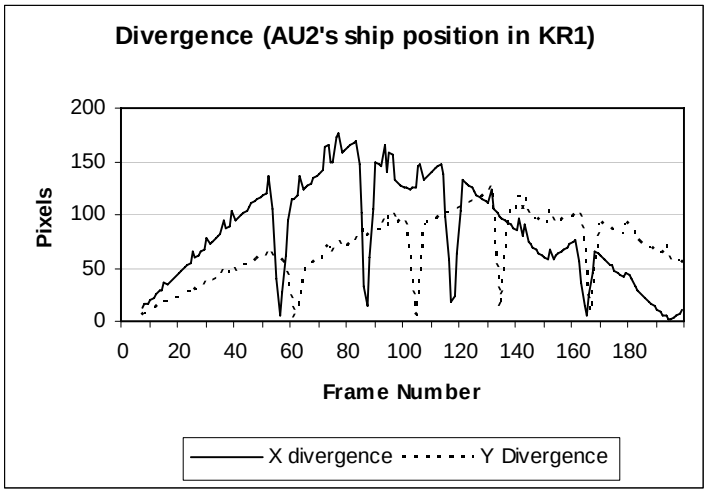
Group G4-3: (Divergence)

The real objects' positions for nodes in the group G4-3 are not presented here due to its appearance in the previous sections. To present the effect of unsynchronised game start time, the case of divergence measurement between node AU2 and node KR1 is displayed here. The FPS values of node AU2 and KR1 are 21.32 and 21.33, respectively as those are displayed in Table 5.16. Therefore, the divergence of real and ghost objects in the two nodes should be neglectable. However, from Table 5.17, node AU2-KR1 pair's divergence values are distinguishably different from the node KR1-AU2 pair. It is caused by the unsynchronised game starting time. When all

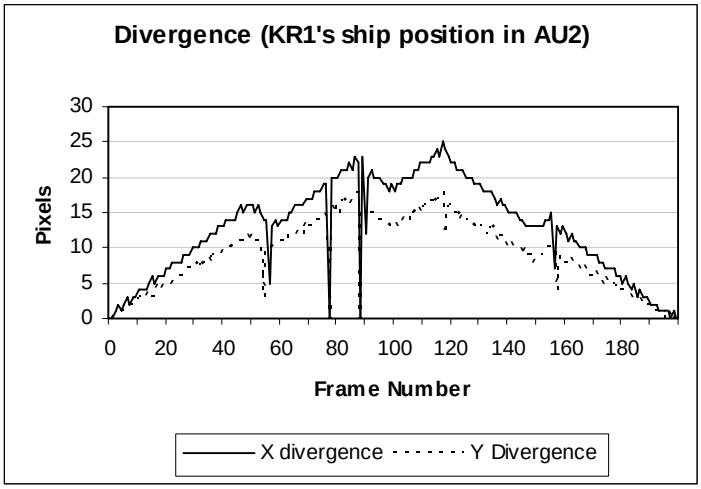
nodes connect to a game owner node and the game is ready to start, the game owner node broadcast to other nodes at the start of the game. Due to the difference of propagation speed of the packet, the unsynchronisation of game start time is unavoidable unless well-established time synchronisation mechanism is run. The divergence graphs of AU2-KR1 and KR1-AU2 pairs are presented in Figure 5.14 (a) and (b), respectively. As can be seen in Figure 5.14 (a), the divergence graph starts from frame 8 due to the packet propagation delay and unsynchronised game starting time.

Table 5.17: Divergence of real and ghost object in Group G4-3

Real Obj.	Ghost Obj.	Div. X (Avg.)	Div. Y (Avg.)	Div. X (STD)	Div. Y (STD)
AU1	AU2	178.05	138.52	0.36	0.00
AU1	KR1	169.32	132.06	1.11	0.21
AU1	KR3	170.69	132.96	1.00	0.22
AU2	AU1	176.81	138.16	0.07	0.07
AU2	KR1	84.76	62.29	1.96	1.43
AU2	KR3	73.54	53.99	3.10	2.09
KR1	AU1	188.05	137.56	0.38	0.23
KR1	AU2	11.71	8.60	5.70	4.36
KR1	KR3	12.19	8.94	0.02	0.02
KR3	AU1	186.05	137.52	0.37	0.45
KR3	AU2	19.91	14.70	0.23	1.07
KR3	KR1	24.09	17.78	0.20	0.18



(a) Position difference between real object of AU2 and ghost object in KR1



(b) Position difference between real object of KR1 and ghost object in AU2

Figure 5.14: Divergence of AU2 and KR1's real and ghost objects

Group G4-3: (Frame Interval)

The average, standard deviation, and median values of frame interval for each node in group G4-3 are shown in Table 5.18. Average frame interval values for each node range from 40.11 ms to 46.92 ms and frame interval is directly related to FPS value of each node, packet drop rate, and packet propagation delay rate. It will be examined in following sections.

Table 5.18: Frame interval in Group G4-3

Destination	Origin	FI AVG	FI STD	FI Median
AU1	AU2	46.90	6.91	48.00
AU1	KR1	46.87	3.50	47.00
AU1	KR3	46.90	9.62	47.00
AU2	AU1	40.11	5.13	40.00
AU2	KR1	46.87	3.13	47.00
AU2	KR3	46.91	9.19	47.00
KR1	AU1	43.50	6.53	46.50
KR1	AU2	46.91	8.30	47.00
KR1	KR3	46.91	10.28	47.00
KR3	AU1	43.52	8.63	46.50
KR3	AU2	46.92	7.98	47.00
KR3	KR1	46.86	8.64	47.00

Group G8-1: (Pre-measured One-Trip Time)

Total eight nodes participate in this group and it is a combination of two groups, group G4-1 and G4-2. The OTT (One-Trip Time) values between nodes in group G4-4 are measured and presented in Table 5.19. It takes 209.95 ms to transfer packets between node AU3 and KR1 and 5.35 ms between node AU1 and AU3. The maximum latency has no effect on the game session which adopts FSR algorithm because every node simply does not concern about other nodes packets for transferring its own packets. However, it has significant effect on game sessions which apply LS and LBS algorithms. It will be covered in following sections.

Table 5.19: One-Time Time between nodes in G8-1

Node No.	AU1	AU2	AU3	AU4	KR1	KR2	KR3	KR4
AU1		9.53	5.35	86.08	203.55	193.80	184.63	102.70
AU2	9.53		8.30	87.48	204.56	197.56	174.69	100.55
AU3	5.35	8.30		88.05	209.95	197.15	198.00	105.05
AU4	86.08	87.49	88.05		125.03	116.28	103.63	177.03
KR1	203.55	204.56	209.95	125.03		75.48	52.95	149.50
KR2	193.80	197.56	197.15	116.28	75.48		49.80	125.30
KR3	184.63	174.69	198.00	103.63	52.95	49.80		118.45
KR4	102.70	100.55	105.05	177.03	149.50	125.30	118.45	

Group G8-1: (Overall Performance)

The overall system performance, packet drop rate, and packet transfer rate of group G8-1 are presented in Table 5.20. As it can be seen in the table, node AU1 shows almost upper benchmark FPS value which is 24.93, which is similar to the experimental results in group G4-1 and G4-3. One thing that is noticeable from the experimental results of group G4-2 is the packet drop rates of node KR2, which jumps from 0.02% to 0.67%. Node KR2 utilises wireless network as it is mentioned in Chapter 4. Due to space and time constraint, and the irrelevance of the topic, the

analysis of the cause of this case will be excluded. However, the effect of packet drop rate will be covered in the following sub-sections.

Table 5.20: FPS, PPS, Drop Rate in G8-1 with FSR

Node	FPS:	Drop Rate (IN)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	24.93	0.04%	139.83	174.57	314.40
AU2	21.33	0.03%	143.45	149.39	292.85
AU3	21.13	0.04%	143.68	148.03	291.71
AU4	21.32	0.02%	143.51	149.37	292.89
KR1	21.33	0.05%	142.94	149.41	292.36
KR2	16.16	0.67%	146.42	113.24	259.66
KR3	21.31	0.06%	142.97	149.27	292.24
KR4	17.20	0.05%	147.07	120.49	267.56

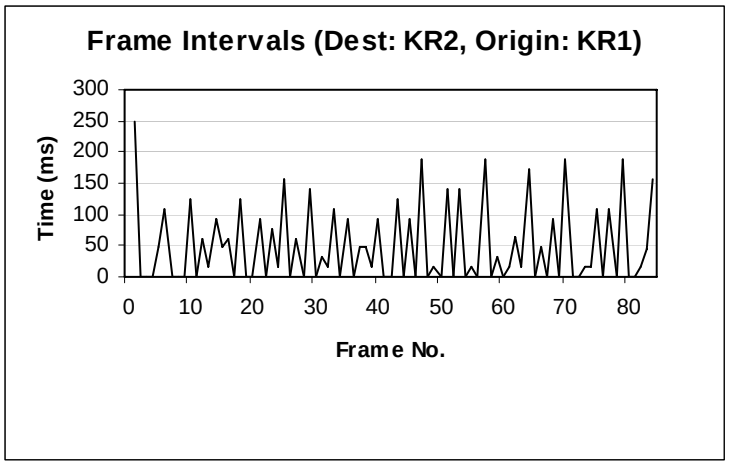
Group G8-1: (Divergence)

All of the trajectory graphs of real objects' position in group G8-1 are presented in previous sections. Table 5.21 displays the divergence of real and ghost objects in group G8-1. As shown in the table, node AU2-AU4 and AU4-AU2 pairs present well-synchronised divergence of real and ghost objects in each node. Unsynchronised game start time effect can be seen from node AU4-KR1 and KR1-AU4 pairs as explained in previous sections.

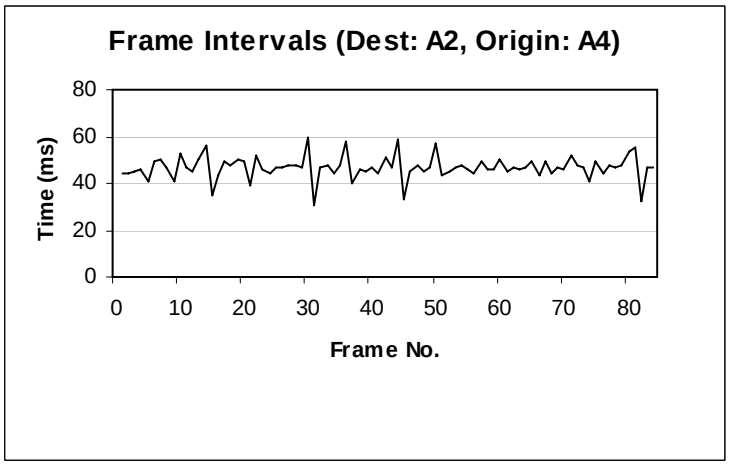
Group G8-1: (Frame Interval)

The average, standard deviation, and median values of frame interval for each node in group G8-1 are shown in Table 5.22. Maximum frame interval (average) value is 61.67 ms between node AU4-KR2 pair and maximum frame interval (standard deviation) value is 62.96 ms between node KR2-KR3 pair. On one hand, Node KR2 records 0.67% of packet drop rate and the frame interval table also shows the KR2 node performs poorly in terms of the regularity of packet transfer. On the other hand, node AU2-AU4 pair performed well in terms of the regularity of packet transfer, which it shows the frame interval value (standard deviation) as 4.26 ms. Figure 5.15

(a) and (b) show the frame interval graphs of node KR2-KR3 and AU2-AU4 pairs, respectively.



(a) Frame Interval of node KR1 in node KR2



(b) Frame Interval of node AU4 in node AU2

Figure 5.15: Frame Interval of node KR1 and AU4

Table 5.21: Divergence of real and ghost object in Group G8-1

Real Obj.	Ghost Obj.	Div. X (Avg.)	Div. Y (Avg.)	Div. X (STD)	Div. Y (STD)
AU1	AU2	174.06	135.71	125.16	103.87
AU1	AU3	182.09	145.69	132.25	103.08
AU1	AU4	174.23	136.00	125.20	103.78
AU1	KR1	166.56	130.36	127.74	103.69
AU1	KR2	214.14	137.80	148.39	107.63
AU1	KR3	166.58	130.42	127.75	103.79
AU1	KR4	191.98	155.12	139.38	112.10
AU2	AU1	174.84	136.64	128.56	100.50
AU2	AU3	86.05	63.45	43.31	29.88
AU2	AU4	0.58	0.41	1.43	0.96
AU2	KR1	86.85	63.70	38.10	24.31
AU2	KR2	197.02	151.90	134.44	104.40
AU2	KR3	79.18	58.14	33.83	21.30
AU2	KR4	195.59	147.67	141.17	101.12
AU3	AU1	181.73	131.83	139.47	102.49
AU3	AU2	94.48	69.10	50.87	39.02
AU3	AU4	94.19	68.89	50.75	38.89
AU3	KR1	153.85	112.36	81.26	62.08
AU3	KR2	183.46	134.45	129.35	96.29
AU3	KR3	153.48	112.08	81.10	61.99
AU3	KR4	187.16	137.28	136.50	101.05
AU4	AU1	184.88	137.00	133.01	97.89
AU4	AU2	12.02	8.91	3.28	4.14
AU4	AU3	85.04	61.80	39.01	36.12
AU4	KR1	88.05	64.71	27.54	32.51
AU4	KR2	181.97	135.38	144.92	95.16
AU4	KR3	78.20	57.54	23.72	28.48
AU4	KR4	192.87	140.31	135.21	107.54
KR1	AU1	192.87	140.31	135.21	107.54
KR1	AU2	9.63	8.87	4.98	3.47
KR1	AU3	75.44	62.48	50.06	30.52
KR1	AU4	0.07	0.04	1.05	0.65
KR1	KR2	217.03	137.42	155.31	92.33
KR1	KR3	11.87	8.77	4.56	5.80
KR1	KR4	208.01	135.95	151.62	97.27
KR2	AU1	192.76	147.87	154.58	99.02
KR2	AU2	214.42	138.40	154.25	105.31
KR2	AU3	215.14	139.30	158.84	99.01
KR2	AU4	214.32	138.43	154.25	105.23
KR2	KR1	214.43	138.33	154.04	105.47
KR2	KR3	214.31	138.39	154.36	105.21
KR2	KR4	156.97	125.53	154.94	102.84
KR3	AU1	182.69	133.83	137.09	102.40
KR3	AU2	12.72	9.36	4.74	3.44
KR3	AU3	77.40	57.17	37.94	27.83
KR3	AU4	12.50	9.21	4.35	3.18
KR3	KR1	12.95	9.56	5.24	3.83
KR3	KR2	187.08	139.83	138.85	101.78
KR3	KR4	193.50	142.19	129.45	96.87
KR4	AU1	198.81	151.18	146.48	103.05
KR4	AU2	180.44	133.37	140.59	93.11
KR4	AU3	185.41	134.41	136.09	94.65
KR4	AU4	185.97	134.93	136.05	95.34
KR4	KR1	186.00	134.52	135.95	94.80
KR4	KR2	143.78	123.54	131.64	102.20
KR4	KR3	185.47	134.42	136.30	94.53

Table 5.22: Frame interval in Group G8-1

Destination	Origin	FI AVG	FI STD	FI Median
AU1	AU2	46.88	7.43	47.00
AU1	AU3	47.31	14.74	46.50
AU1	AU4	46.88	6.06	46.50
AU1	KR1	54.20	25.39	48.00
AU1	KR2	54.47	29.58	46.50
AU1	KR3	51.50	22.21	48.50
AU1	KR4	52.43	21.52	50.50
AU2	AU1	40.12	5.19	40.00
AU2	AU3	47.31	14.77	47.00
AU2	AU4	46.89	4.26	47.00
AU2	KR1	54.14	23.28	47.50
AU2	KR2	54.47	27.64	48.00
AU2	KR3	52.61	22.46	52.00
AU2	KR4	52.42	20.57	51.50
AU3	AU1	40.13	14.86	46.00
AU3	AU2	46.87	15.11	47.00
AU3	AU4	46.88	13.20	47.00
AU3	KR1	54.17	28.57	47.00
AU3	KR2	54.47	31.58	47.00
AU3	KR3	52.59	28.23	54.50
AU3	KR4	52.43	27.44	54.50
AU4	AU1	40.12	9.39	46.00
AU4	AU2	46.88	10.90	47.00
AU4	AU3	47.31	14.42	47.00
AU4	KR1	46.87	5.94	47.00
AU4	KR2	61.67	45.65	47.00
AU4	KR3	46.92	12.75	47.00
AU4	KR4	58.14	37.56	62.00
KR1	AU1	40.12	8.85	46.50
KR1	AU2	46.91	9.62	47.00
KR1	AU3	47.31	15.86	47.00
KR1	AU4	54.16	27.49	47.00
KR1	KR2	52.64	25.88	47.00
KR1	KR3	52.61	23.36	54.50
KR1	KR4	52.44	21.65	54.50
KR2	AU1	40.49	55.95	0.00
KR2	AU2	47.37	58.33	16.00
KR2	AU3	47.96	59.01	16.00
KR2	AU4	47.37	58.40	16.00
KR2	KR1	47.38	58.85	16.00
KR2	KR3	53.20	62.96	16.00
KR2	KR4	52.94	62.30	23.50
KR3	AU1	40.12	7.30	40.00
KR3	AU2	46.91	6.30	48.00
KR3	AU3	47.33	15.50	46.50
KR3	AU4	46.88	5.15	47.00
KR3	KR1	54.17	24.58	48.00
KR3	KR2	60.18	41.76	54.00
KR3	KR4	52.42	22.07	52.50
KR4	AU1	40.12	42.60	16.00
KR4	AU2	46.91	43.51	47.00
KR4	AU3	47.31	45.22	46.50
KR4	AU4	46.89	43.29	46.50
KR4	KR1	54.19	51.11	54.50
KR4	KR2	54.47	53.11	54.50
KR4	KR3	46.84	43.12	47.00

5.3.3. Experimental results on Duel-X with LS

Group G4-1: (Pre-measured One-Trip Time)

As it is mentioned in previous sections, one way trip time is measured between nodes in experimental groups before each game session starts. Table 5.23 shows the OTT values among nodes in group G4-1 which is measured ten times and averaged. The maximum difference between this average values and the values in FSR sessions is 0.665 ms, which is neglectable especially for LAN-based game sessions. The OTT values here will be used in packet retransmission schemes which will be explained fully in following sections. RTT (Round-Trip Time) values which are measured during game sessions will be also presented and comparison between these two values is presented in the following sub-sections.

Table 5.23: One-Time Time between nodes in G4-1

Node No.	AU1	AU2	AU3	AU4
AU1		7.86	5.50	6.19
AU2	7.86		7.34	9.71
AU3	5.50	7.34		4.16
AU4	6.19	9.71	4.16	

Group G4-1: (Overall Performance)

Experimental parameters are displayed in Table 5.24. The experiments in this sessions utilises LS as a synchronisation algorithm, four total players, and 40ms of frame interval, no transmission scheme, and no aggregation.

Table 5.24: Parameters for Group G4-1

P2P Type:	Mesh
Synch Algorithm:	LS
Transmission:	None
Aggregation:	no
Frame Interval (ms):	40
Playout Delay (ms):	
Total Players:	4

One more item is added to overall performance table with FPS (frame per second), PPS (Packets per second), and drop rate values, which is packet resending rate and these values are presented in Table 5.25, generated from the experiments of group G4-1 with LS algorithm. Different from FSR algorithm which does not care about other nodes' situation, LS algorithm adopts a "send-and-wait" scheme. As it can be seen in the table, every node performs identically in terms of game execution speed as denoted in FPS values. The drop rates are similar to the experimental results of the case that adopts FSR algorithm with the same group G4-1. However, PPS (IN) and PPS (OUT) values become double due to acknowledgement packets. The comparison of PPS values will be covered fully in following analysis section.

Table 5.25: FPS, PPS, Drop Rate in G4-1 with LS

Node	FPS:	Drop Rate (IN)	Resend Rate (OUT)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	19.88	0.00%	1.50%	121.15	121.16	242.31
AU2	19.88	0.01%	2.55%	122.42	122.44	244.86
AU3	19.88	0.01%	2.69%	122.73	122.75	245.48
AU4	19.88	0.00%	3.64%	123.97	123.99	247.96

Group G4-1: (Divergence)

Divergence comparison between nodes is meaningless in LS and LBS algorithms because these algorithms maintain perfect consistency due to their "send-and-wait" scheme. Therefore, this section will be out of this thesis from now on.

Group G4-1: (Frame Interval & RTT)

The average and standard deviation values of frame interval for each node are shown in Table 5.26. As it can be seen in the table, average frame interval value for each node is approximately 50.20 ms and it reflects well the game execution speed that is 19.88 fps. Figure 5.16 (a), (b), and (c) show frame interval graphs of node AU2, AU3,

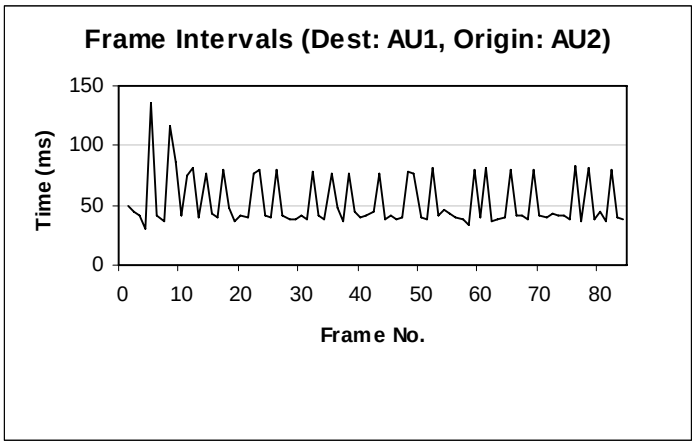
and AU3 which are measured in node AU1. These graphs demonstrate the characteristics of LS algorithm, synchronised game execution. As it can be seen in the figures, the shapes of graphs are almost identical.

Table 5.26: Frame interval in Group G4-1 with LS algorithm

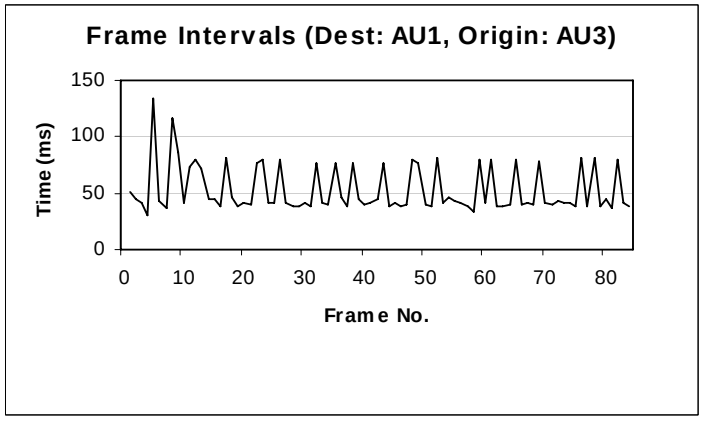
Destination	Origin	FI AVG	FI STD
AU1	AU2	50.19	18.17
AU1	AU3	50.18	18.23
AU1	AU4	50.20	18.25
AU2	AU1	50.21	13.28
AU2	AU3	50.21	13.31
AU2	AU4	50.21	13.30
AU3	AU1	50.21	16.03
AU3	AU2	50.18	16.83
AU3	AU4	50.20	17.20
AU4	AU1	50.22	12.50
AU4	AU2	50.22	12.50
AU4	AU3	50.22	12.50

In addition to frame interval measurement, RTT values are calculated in game sessions that utilises LS and LBS algorithms. When game packets are sent from origin nodes to destination nodes, acknowledgement packets that contain frame numbers of the game packets are sent to the origin nodes. The departure time of game packets and the arrival time of correspondent acknowledgement packets are recorded in external files for further analysis. The measured RTT average and standard deviation values are displayed in Table 5.27.

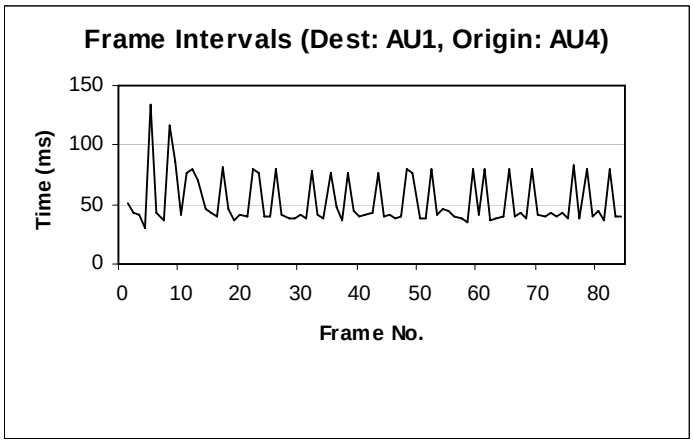
To examine the symmetry of RTT between two nodes in LAN environment Figure 5.17 (a) and (b) display RTT graphs between node AU1-AU2 and AU2-AU1 pairs, respectively. Two graphs show identical height of pike at frame 13 which is the value of 35 ms. The general shapes of two graphs are similar to each other, but average RTT value for node AU1 is 3.49 ms which is smaller than that of node AU0 which is 5.58 ms in case of the experiment. The effect of RTT values on game execution speed with FSR, LS, and LBS algorithms will be examined in following analysis sections.



(a) Frame interval of node AU2 measured in node AU1



(b) Frame interval of node AU2 measured in node AU1

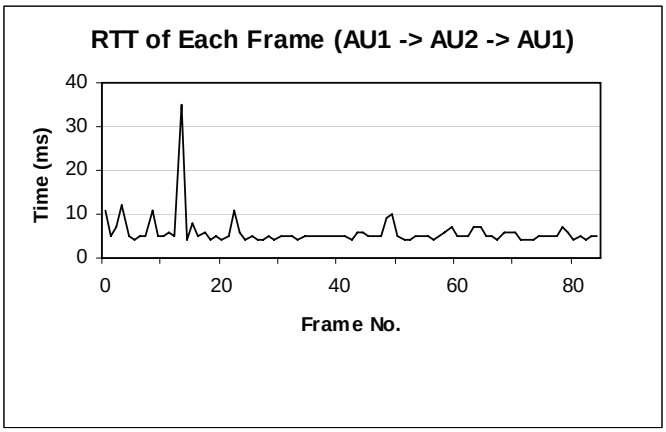


(c) Frame interval of node AU2 measured in node AU1

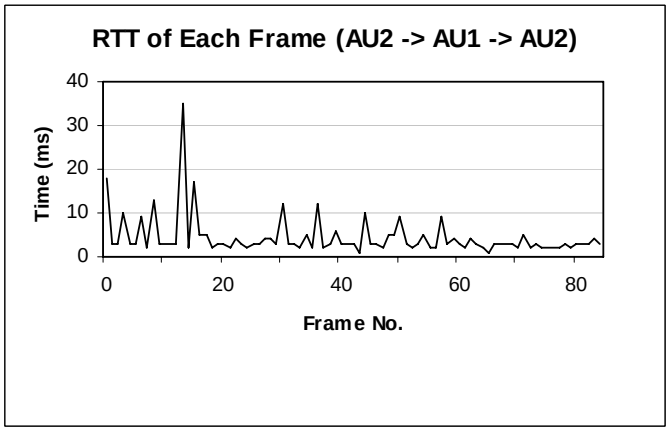
Figure 5.16: Frame Interval of node AU2, AU3, and AU4 measured in node AU1

Table 5.27: RTT values in Group G4-1 with LS algorithm

Destination	Origin	RTT AVG	RTT STD
AU1	AU2	5.66	2.39
AU1	AU3	7.46	7.60
AU1	AU4	4.98	2.47
AU2	AU1	3.63	2.67
AU2	AU3	6.85	7.94
AU2	AU4	4.90	3.06
AU3	AU1	1.23	5.15
AU3	AU2	1.05	4.30
AU3	AU4	0.94	3.89
AU4	AU1	0.67	2.96
AU4	AU2	0.91	3.65
AU4	AU3	4.64	9.17



(a) RTT graph of node AU1



(b) RTT graph of node AU2

Figure 5.17: RTT values between node AU1 and AU2

Group G4-2: (Pre-measured One-Trip Time)

The pre-measured OTT values between nodes in group G4-2 with LS algorithm is displayed in Table 5.28. Compare to the results on OTT values in the same group, which is G4-2 but with FSR in the previous section, node KR2's RTT values with other nodes are about 10% ~ 30% higher in this case. As mentioned previously, node KR2 utilises wireless network and the node show irregular network performance compared to other nodes which adopt wired network. This thesis, however, concentrates on the effect of OTT and packet resending rate values on game execution speed and bandwidth requirement. These analyses will be provided in following sections.

Table 5.28: One-Time Time between nodes in G4-2 with LS algorithm

Node No.	KR1	KR2	KR3	KR4
KR1		46.93	9.35	27.73
KR2	46.93		55.45	52.93
KR3	9.35	55.45		29.53
KR4	27.73	52.93	29.53	

Group G4-2: (Overall Performance)

Table 5.29 shows FPS, inbound packet drop rate, outbound packet resending rate, inbound, outbound, and total packets per second values for nodes in the group G4-2 with LS algorithm. Shown in the table, the packet resending rates of all nodes are much higher than those from group G4-1. It is caused by pre-measured OTT values which are significantly different from RTT values which are measured during game play. Also, Duel-X does not adopt past packet prevention mechanism which filters past frame packets. The past packet prevention mechanism is implemented in COMP2P but not in Duel-X for the comparison between them. With the past packet prevention mechanism, the PPS (OUT) value for node KR1 should be similar to 43.68

with 0% packet drop and delay, however, it increases to 50.15 because of both not only packet drop and delay but also past frame packets.

Table 5.29: FPS, PPS, Drop Rate in G4-2 with LS

Node	FPS:	Drop Rate (IN)	Resend Rate (OUT)	PPS (IN)	PPS (OUT)	PPS (ALL)
KR1	7.28	0.04%	12.77%	50.15	50.15	100.30
KR2	7.24	0.00%	21.54%	55.61	55.65	111.27
KR3	7.28	0.00%	15.71%	51.94	51.95	103.89
KR4	7.29	0.00%	20.01%	54.65	54.70	109.34

Group G4-2: (Frame Interval & RTT)

Frame interval values in group G4-2 is presented in Table 5.30. As it can be seen in the table, frame interval values for each node is about 137 ms which reflects game execution speed of each node (about 7.2 fps) well. Frame interval value for each node is directly affected by the RTT value of the slowest node in the group that utilises LS algorithm mentioned in the previous sections. The RTT values for all nodes in the group G4-2 are shown in Table 5.31 and the relationship between RTT (or OTT) values and game execution speed of LS algorithm is analysed in following sections.

Table 5.30: Frame interval in Group G4-2 with LS algorithm

Destination	Origin	FI AVG	FI STD
KR1	KR2	137.04	52.89
KR1	KR3	137.04	52.89
KR1	KR4	137.04	52.89
KR2	KR1	137.95	71.93
KR2	KR3	137.95	71.87
KR2	KR4	137.95	71.88
KR3	KR1	137.06	54.51
KR3	KR2	137.06	54.59
KR3	KR4	137.06	54.65
KR4	KR1	137.27	85.94
KR4	KR2	137.25	86.01
KR4	KR3	137.25	86.09

Table 5.31: RTT values in Group G4-2 with LS algorithm

Destination	Origin	RTT AVG	RTT STD
KR1	KR2	88.70	30.76
KR1	KR3	3.67	8.56
KR1	KR4	64.45	34.35
KR2	KR1	80.95	26.52
KR2	KR3	92.38	32.98
KR2	KR4	141.15	48.78
KR3	KR1	3.39	7.47
KR3	KR2	104.36	31.07
KR3	KR4	74.79	33.46
KR4	KR1	87.39	38.78
KR4	KR2	133.91	45.58
KR4	KR3	96.01	34.87

Group G4-3: (Pre-measured One-Trip Time)

Table 5.32 shows OTT values between nodes in group G4-3 with LS algorithm. The values are very similar to the values from the group with FSR and the RTT which are measured during the game sessions shown in the following section.

Table 5.32: One-Time Time between nodes in G4-3 with LS

Node No.	AU1	AU2	KR1	KR3
AU1		7.70	164.95	176.40
AU2	7.70		168.00	170.33
KR1	164.95	168.00		13.75
KR3	176.40	170.33	13.75	

Group G4-3: (Overall Performance)

The overall performance values such as fps, packet drop rate (inbound), PPS (inbound, outbound, and total) are shown in Table 5.33. The average FPS value in the game session is about 4.8 and it indicates that OTT value of the slowest node in this group is approximately 208.3 ms. This value is slightly larger than the largest OTT value of the nodes in group G4-3. It will be explained with RTT values which are measured during the game sessions, the standard deviation of the RTT values are listed in the following sections.

Table 5.33: FPS, PPS, Drop Rate in G4-3 with LS

Node	FPS:	Drop Rate (IN)	Resend Rate (OUT)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	4.80	0.00%	0.54%	28.96	29.01	57.96
AU2	4.81	0.00%	0.51%	28.98	29.05	58.03
KR1	4.78	0.00%	0.43%	28.86	28.90	57.76
KR3	4.79	0.00%	0.51%	28.89	28.93	57.83

Group G4-3: (Frame Interval & RTT)

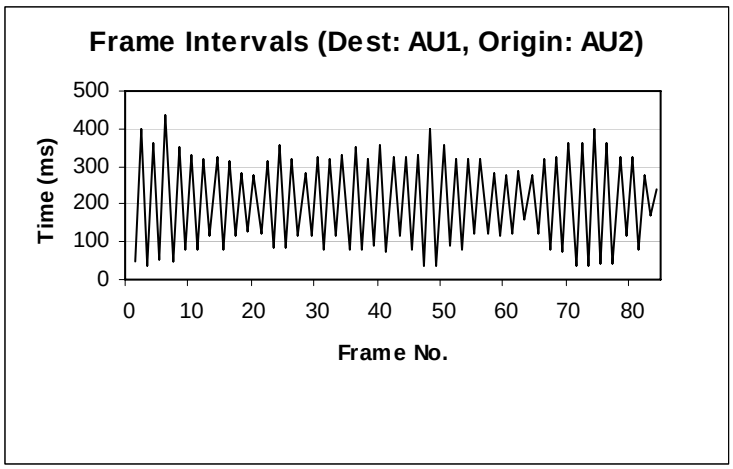
As it is mentioned in previous sections, group G4-3 consists of four nodes, two in South Korea and two in Australia. Even though the measured RTT values during the game sessions between AU1-AU2 and KR1-KR3 pairs are less than 10 ms, shown in Table 5.35 and the average frame interval values among nodes in this group are approximately 208 ms. Figure 5.18 (a) and (b) give the frame interval values of node AU2, which are measured in node AU1 and the RTT values between node AU1 and AU2.

Table 5.34: Frame interval in Group G4-3 with LS algorithm

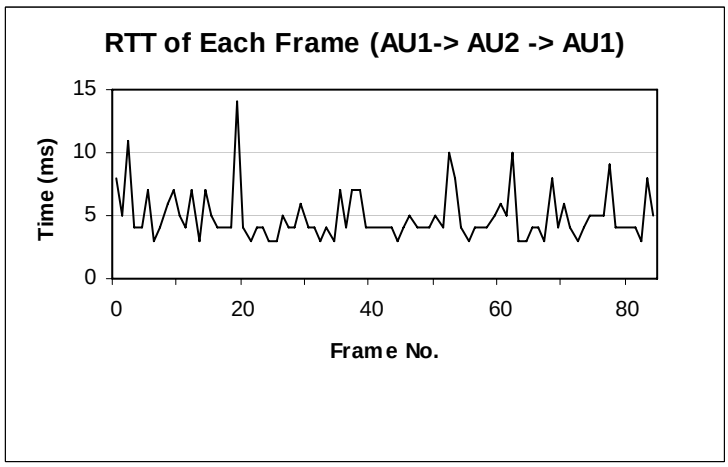
Destination	Origin	FI AVG	FI STD
AU1	AU2	208.04	125.35
AU1	KR1	208.04	125.62
AU1	KR3	208.04	125.55
AU2	AU1	208.09	128.43
AU2	KR1	208.09	128.46
AU2	KR3	208.09	128.53
KR1	AU1	208.60	127.81
KR1	AU2	208.60	127.81
KR1	KR3	208.60	127.81
KR3	AU1	208.51	126.97
KR3	AU2	208.51	127.02
KR3	KR1	208.51	127.23

Table 5.35: RTT values in Group G4-3 with LS algorithm

Destination	Origin	RTT AVG	RTT STD
AU1	AU2	5.20	3.00
AU1	KR1	328.03	23.45
AU1	KR3	344.85	25.34
AU2	AU1	3.53	3.12
AU2	KR1	337.01	28.41
AU2	KR3	329.43	28.16
KR1	AU1	327.14	14.28
KR1	AU2	328.49	15.71
KR1	KR3	4.04	8.36
KR3	AU1	345.31	27.39
KR3	AU2	329.17	17.20
KR3	KR1	8.07	9.45



(a) Frame interval of node AU2 measured in node AU1



(b) RTT graph of node AU1

Figure 5.18: Frame interval and RTT graphs of node AU1 & AU2

Group G8-1: (Pre-measured One-Trip Time)

Again, echo-requesting packets are propagated to all nodes five times before game sessions start and arrived acknowledge packets are used to calculate average RTT times between each node. These values are sent to a game session creator and a unified OTT table shown in Table 5.36 is sent to all other nodes and used for packet retransmission schemes in LS and LBS algorithms.

Table 5.36: One-Time Time between nodes in G8-1

Node No.	AU1	AU2	AU3	AU4	KR1	KR2	KR3	KR4
AU1		8.13	5.50	7.23	165.05	213.18	196.53	190.33
AU2	8.13		7.25	8.98	167.53	222.50	189.13	182.50
AU3	5.50	7.25		2.00	177.75	213.70	196.98	197.28
AU4	7.23	8.98	2.00		164.00	225.80	192.90	183.68
KR1	165.05	167.53	177.75	164.00		73.40	30.03	32.38
KR2	213.18	222.50	213.70	225.80	73.40		92.83	67.18
KR3	196.53	189.13	196.98	192.90	30.03	92.83		50.73
KR4	190.33	182.50	197.28	183.68	32.38	67.18	50.73	

Group G8-1: (Overall Performance)

The FPS, PPS, drop rates of nodes in group G8-1 is shown in Table 5.37. As it can be seen in the table, the overall FPS value is 3.45 and this value indicates that the frame interval is about 288 ms. The inbound and outbound of PPS values denote that node KR2 performs poorly compared to other nodes mainly due to its high packet resending rate which is 11.18%. The main factor of the high packet resending rate of node KR2 is caused by the difference between pre-measured OTT values and the RTT values that are measured during game sessions. To verify this fact in detail, two tables are presented, Table 5.38 and x4. In the Table 5.38, greyed cells indicate that the OTT values between two nodes are under-measured, which means the pre-measured OTT values are smaller than the OTT values that are measured during game session. LS and LBS algorithms resend packets if expected acknowledge packet does not arrive in

time then another packet is sent. The waiting time is 110% of pre-measured OTT time. As it can be seen in the table, most grey cells are located in node KR2's row and column.

Table 5.37: FPS, PPS, Drop Rate in G8-1 with LS

Node	FPS:	Drop Rate (IN)	Resend Rate (OUT)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	3.45	0.00%	2.27%	49.57	49.67	99.24
AU2	3.45	0.00%	2.78%	49.82	49.93	99.76
AU3	3.45	0.00%	2.06%	49.46	49.56	99.03
AU4	3.45	0.07%	2.71%	49.78	49.88	99.66
KR1	3.44	0.00%	4.61%	50.60	50.67	101.27
KR2	3.44	0.00%	11.18%	54.16	54.35	108.51
KR3	3.44	0.00%	6.67%	51.73	51.84	103.57
KR4	3.45	0.00%	6.45%	51.61	51.71	103.32

Also, the fluctuations of OTT values are measured in forms of the standard deviation of OTT values between nodes in group G8-1 and these values are presented in

Table 5.39. As it can be seen in the table, the OTT STD values for node KR2 is generally higher than ones of other nodes' and these are marked in grey.

Table 5.38: OTT value difference between pre-measured and in-session values

Node No.	AU1	AU2	AU3	AU4	KR1	KR2	KR3	KR4
AU1		-4.10	-2.98	-6.29	-8.43	7.61	-10.69	-3.20
AU2	-1.80		-4.20	-8.68	-3.97	4.07	-10.77	-4.24
AU3	0.98	-3.56		-1.13	-13.06	9.67	-9.19	-7.13
AU4	-2.08	-5.76	0.19		-0.70	1.98	-14.09	-3.41
KR1	-4.51	0.62	-8.36	-0.62		9.55	-10.67	8.72
KR2	8.62	6.06	10.14	3.54	0.95		-2.65	19.96
KR3	-11.19	-11.34	-8.81	-14.93	-13.68	4.57		3.19
KR4	-3.83	-2.89	-5.16	-5.05	-5.64	28.40	-2.39	

Table 5.39: OTT standard deviation values between nodes in group G8-1 with LS

Node No.	AU1	AU2	AU3	AU4	KR1	KR2	KR3	KR4
AU1		6.11	9.65	5.30	22.71	67.60	27.70	44.47
AU2	4.13		9.05	3.04	22.99	67.02	26.53	42.80
AU3	9.11	7.75		6.73	24.18	67.99	28.30	43.32
AU4	3.84	4.65	7.98		22.78	67.11	26.50	43.06
KR1	22.54	23.58	24.89	22.84		54.47	8.78	34.48
KR2	56.28	57.57	54.79	56.51	45.45		49.58	57.85
KR3	27.40	26.63	28.98	27.15	10.18	51.98		32.05
KR4	40.97	41.44	42.33	40.59	30.85	62.19	34.80	

Group G8-1: (Frame Interval)

As it is calculated using FPS values of nodes in group G8-1, the frame interval values are about 288 ms and Table 5.41 also shows the standard deviation values of the frame interval. For completeness, the average and standard deviation RTT values are presented in Table 5.42.

5.3.4. Experimental results on Duel-X with LBS

Group G4-1: (Pre-measured One-Trip Time)

Group G4-1 consists of four nodes that are resided in LAN and pre-measured OTT values are less than 10 ms as it can be seen in Table 5.40. The pre-defined frame interval and packet transfer time gap is 40 ms. Therefore, the game execution speed of each node should be upper benchmark speed unless there is significant packet delay and/or drop. Next section will present overall performance of each node and explanation for each value.

Table 5.40: One-Time Time between nodes in G4-1

Node No.	AU1	AU2	AU3	AU4
AU1		7.60	4.28	5.16
AU2	7.60		5.84	9.52
AU3	4.28	5.84		4.17
AU4	5.16	9.52	4.17	

Table 5.41: Frame interval in Group G8-1 with LS

Destination	Origin	FI AVG	FI STD
AU1	AU2	288.57	118.52
AU1	AU3	288.57	118.59
AU1	AU4	288.57	118.88
AU1	KR1	288.57	119.54
AU1	KR2	288.57	119.21
AU1	KR3	288.57	119.39
AU1	KR4	288.57	119.34
AU2	AU1	288.56	121.10
AU2	AU3	288.56	121.14
AU2	AU4	288.56	121.18
AU2	KR1	288.56	121.24
AU2	KR2	288.56	121.14
AU2	KR3	288.56	121.21
AU2	KR4	288.56	121.15
AU3	AU1	288.57	118.63
AU3	AU2	288.57	118.69
AU3	AU4	288.76	118.58
AU3	KR1	288.57	118.34
AU3	KR2	288.65	118.29
AU3	KR3	288.65	118.36
AU3	KR4	288.65	118.65
AU4	AU1	288.57	120.83
AU4	AU2	288.57	120.83
AU4	AU3	288.57	120.83
AU4	KR1	288.50	120.92
AU4	KR2	288.50	120.92
AU4	KR3	288.50	120.92
AU4	KR4	288.50	120.74
KR1	AU1	289.40	72.87
KR1	AU2	289.40	72.87
KR1	AU3	289.40	72.87
KR1	AU4	289.40	72.87
KR1	KR2	289.40	72.87
KR1	KR3	289.40	72.87
KR1	KR4	289.40	72.87
KR2	AU1	290.73	141.28
KR2	AU2	290.73	141.39
KR2	AU3	290.73	141.66
KR2	AU4	290.73	141.61
KR2	KR1	290.73	141.49
KR2	KR3	290.73	141.46
KR2	KR4	290.73	141.28
KR3	AU1	289.46	82.89
KR3	AU2	289.46	82.89
KR3	AU3	289.46	82.78
KR3	AU4	289.46	82.97
KR3	KR1	289.46	82.78
KR3	KR2	289.46	82.58
KR3	KR4	289.46	82.66
KR4	AU1	289.36	106.25
KR4	AU2	289.36	106.25
KR4	AU3	289.36	106.41
KR4	AU4	289.36	106.55
KR4	KR1	289.36	106.63
KR4	KR2	289.36	106.82
KR4	KR3	289.36	106.82

Table 5.42: RTT values in Group G8-1 with LS

Destination	Origin	RTT AVG	RTT STD
AU1	AU2	12.66	4.13
AU1	AU3	12.95	9.11
AU1	AU4	10.29	3.84
AU1	KR1	321.09	22.54
AU1	KR2	443.60	56.28
AU1	KR3	370.67	27.40
AU1	KR4	373.00	40.97
AU2	AU1	8.05	6.11
AU2	AU3	7.37	7.75
AU2	AU4	6.43	4.65
AU2	KR1	336.29	23.58
AU2	KR2	457.13	57.57
AU2	KR3	355.58	26.63
AU2	KR4	359.21	41.44
AU3	AU1	5.04	9.65
AU3	AU2	6.10	9.05
AU3	AU4	4.37	7.98
AU3	KR1	338.78	24.89
AU3	KR2	447.68	54.79
AU3	KR3	376.35	28.98
AU3	KR4	384.24	42.33
AU4	AU1	1.87	5.30
AU4	AU2	0.61	3.04
AU4	AU3	1.73	6.73
AU4	KR1	326.76	22.84
AU4	KR2	458.68	56.51
AU4	KR3	355.93	27.15
AU4	KR4	357.25	40.59
KR1	AU1	313.23	22.71
KR1	AU2	327.13	22.99
KR1	AU3	329.39	24.18
KR1	AU4	326.60	22.78
KR1	KR2	148.70	45.45
KR1	KR3	32.69	10.18
KR1	KR4	53.49	30.85
KR2	AU1	441.59	67.60
KR2	AU2	453.13	67.02
KR2	AU3	446.73	67.99
KR2	AU4	455.55	67.11
KR2	KR1	165.90	54.47
KR2	KR3	194.81	51.98
KR2	KR4	191.17	62.19
KR3	AU1	371.68	27.70
KR3	AU2	356.72	26.53
KR3	AU3	375.58	28.30
KR3	AU4	357.62	26.50
KR3	KR1	38.72	8.78
KR3	KR2	180.37	49.58
KR3	KR4	96.68	34.80
KR4	AU1	374.26	44.47
KR4	AU2	356.53	42.80
KR4	AU3	380.30	43.32
KR4	AU4	360.53	43.06
KR4	KR1	82.19	34.48
KR4	KR2	174.28	57.85
KR4	KR3	107.83	32.05

Group G4-1: (Overall Performance)

Experimental parameters are displayed in Table 5.43. The experiments in this sessions utilise LBS as a synchronisation algorithm, four total players, and 40ms of frame interval, no transmission scheme, no aggregation, and 160 ms playout delay. Playout delay is the time duration of command generation and its execution time as it is explained in previous sections. The playout delay works like the buffering time of streaming video. In analysis section, the effect of the playout delay will be examined in detail.

Table 5.43: Parameters for Group G4-1 with LBS

P2P Type:	Mesh
Synch Algorithm:	LBS
Transmission:	None
Aggregation:	no
Frame Interval (ms):	40
Playout Delay (ms):	160
Total Players:	4

The overall performance of each node is presented in Table 5.44. As it can be seen in the table, the FPS values of nodes denote perfectly synchronised game play even though there are several resent packets due to packet delay and drop. However, the FPS values are not upper benchmark values which are expected to be 25. This divergence is caused by one of characteristics of LS and LBS algorithms. On one hand, the LS and LBS algorithms synchronise its performance on the slowest nodes in terms of game execution speed and network latency.

On the other hand, FSR algorithm performs best among all three algorithms in terms of game execution speed with the ignorance of other nodes' transmission status. Therefore, the game execution speed of nodes in the same group with FSR indicates each node's best performance in terms of FPS. According to the experimental results

from the same group with FSR in previous sections, node AU3 performs worst which is 21.26 and this value is similar to the FPS values of each node in group G4-1 with LBS.

Table 5.44: FPS, PPS, Drop Rate in G4-1 with LBS

Node	FPS:	Drop Rate (IN)	Resend Rate (OUT)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	21.21	0.03%	1.55%	129.23	129.27	258.50
AU2	21.20	0.03%	2.79%	130.83	130.86	261.70
AU3	21.21	0.05%	2.33%	130.25	130.28	260.53
AU4	21.21	0.01%	3.17%	131.45	131.49	262.95

Group G4-1: (Frame Interval & RTT)

The average and standard deviation values of frame interval for each node are shown in Table 5.45. As shown in the table, average frame interval value for each node is approximately 47.22 ms and it reflects well the game execution speed that is 21.21 fps.

One thing needs to be taken into account is that the standard deviation value of frame interval for each node is lower than its value from the same group with LS and it is similar to the value for the same group with FSR. More detail analysis will be provided in following analysis section. Figure 5.19 depicts the frame interval values of node AU3 which are measured in node AU4 which shows very regular frame interval values except one peak at frame 18.

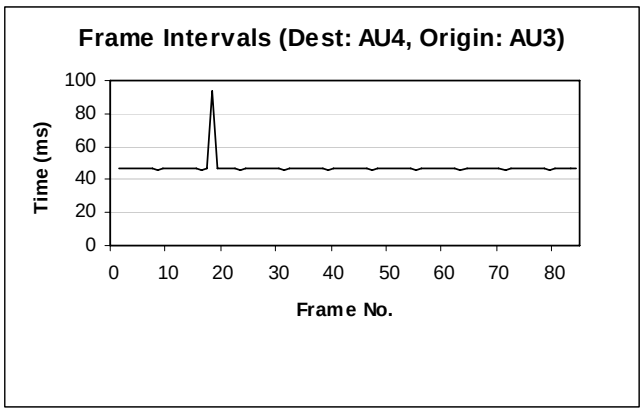


Figure 5.19: Frame Interval of node AU3 measured in node AU4

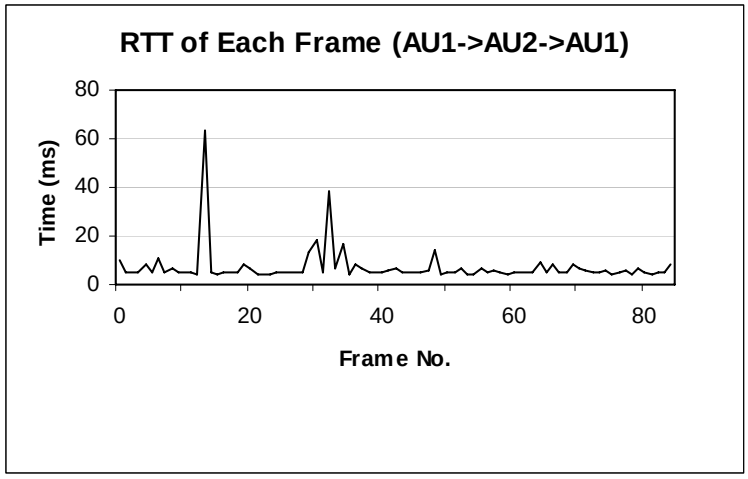
Table 5.45: Frame interval in Group G4-1 with LBS algorithm

Destination	Origin	FI AVG	FI STD
AU1	AU2	47.22	15.83
AU1	AU3	47.22	15.86
AU1	AU4	47.22	15.92
AU2	AU1	47.26	6.32
AU2	AU3	47.26	6.31
AU2	AU4	47.26	6.34
AU3	AU1	47.36	11.93
AU3	AU2	47.36	13.89
AU3	AU4	47.36	14.86
AU4	AU1	47.27	4.27
AU4	AU2	47.26	4.26
AU4	AU3	47.26	4.26

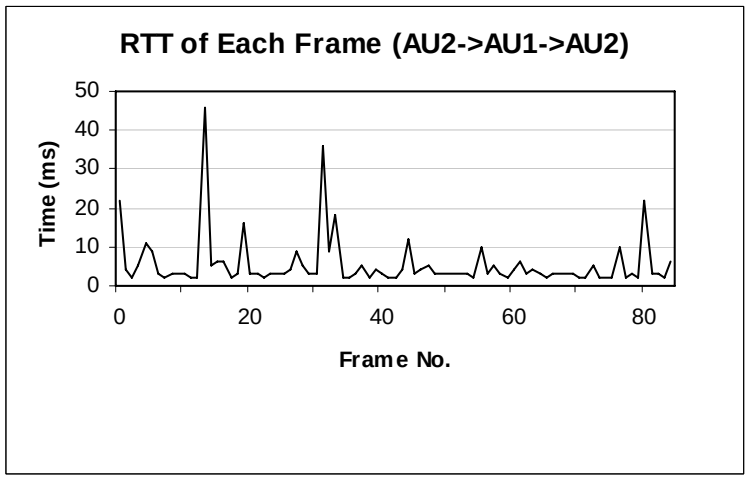
Table 5.46 shows the RTT values between nodes in group G4-1 with LBS algorithm and Figure 5.20 displays the RTT graphs of node AU1 and AU2. As it can be seen in the figure, the two graphs depict similar hight and timing for peaks and it may be originated from the condition of the network which could be affected by hubs, switches, and/or routers rather than the nodes themselves. However, it is not verifiable and omitted due to time and space constraints of the thesis and irrelevance from the topic of this thesis.

Table 5.46: RTT values in Group G4-1 with LBS algorithm

Destination	Origin	RTT AVG	RTT STD
AU1	AU2	5.69	2.64
AU1	AU3	7.84	7.85
AU1	AU4	5.03	2.53
AU2	AU1	3.83	3.08
AU2	AU3	7.41	8.63
AU2	AU4	5.26	3.42
AU3	AU1	2.05	6.54
AU3	AU2	1.75	5.15
AU3	AU4	4.08	7.05
AU4	AU1	0.95	3.87
AU4	AU2	1.32	4.58
AU4	AU3	4.95	9.72



(a) RTT graph of node AU1



(b) RTT graph of node AU2

Figure 5.20: RTT values between node AU1 and AU2

Group G4-2: (Pre-measured One-Trip Time)

The pre-measured OTT values between nodes in group G4-2 with LBS algorithm is presented in Table 5.47. Comparing to the pre-measured OTT values from the same group with FSR and LB algorithms, the values from LBS are generally higher than others. However, the packet resending rates for nodes in group G4-2 with LBS are lower than that with LS. It will be explained in the next section.

Table 5.47: One-Trip Time between nodes in G4-2 with LBS algorithm

Node No.	KR1	KR2	KR3	KR4
KR1		78.05	12.20	27.78
KR2	78.05		88.25	90.30
KR3	12.20	88.25		30.18
KR4	27.78	90.30	30.18	

Group G4-2: (Overall Performance)

The overall performance of group G4-2 with LBS is displayed in Table 5.48. As it can be seen in the table, the FPS values for nodes are well-synchronised, however, the values are not upper benchmark in terms of game execution speed. The longest latency or OTT value between nodes is 90.30 ms in the OTT table and it becomes lower than 70 ms as measured during game sessions. These values are less than pre-defined playout delay which is 160 ms. Therefore, the game execution speed should be upper benchmark speed which is about 25. The difference between expected and measured FPS values is from the slowest node in terms of game execution speed. According to the overall performance results from the same group with FSR, node KR4 records the slowest FPS value which is about 14.32, very simular to the measured FPS values in Table 5.48.

Table 5.48: FPS, PPS, Drop Rate in G4-2 with LBS

Node	FPS:	Drop Rate (IN)	Resend Rate (OUT)	PPS (IN)	PPS (OUT)	PPS (ALL)
KR1	13.80	0.00%	9.02%	91.20	91.26	182.46
KR2	13.73	0.00%	2.82%	84.79	84.90	169.69
KR3	13.81	0.00%	10.65%	93.24	93.29	186.54
KR4	13.79	0.00%	15.45%	98.80	98.89	197.68

Group G4-2: (Frame Interval & RTT)

As explained in previous sections, PPS values are directly proportional to packet resending rate. Node KR4 has the largest PPS value among nodes, because the node has the largest packet resending rate as shown in Table 5.48. Node KR2's packet

resending is caused by delayed acknowledgement packets, the similar cases are examined in the same group with LS algorithm. Most of the pre-measured OTT values except for node KR4 are smaller than the half of RTT values measured during game sessions. The exception causes the high packet resending rate which is 15.45% compared to others. Table 5.49 and Table 5.50 show the frame interval and RTT values between nodes in group G4-2.

Table 5.49: Frame interval in Group G4-2 with LBS algorithm

Destination	Origin	FI AVG	FI STD
KR1	KR2	72.45	35.49
KR1	KR3	72.45	35.49
KR1	KR4	72.45	35.50
KR2	KR1	72.78	41.34
KR2	KR3	72.78	41.32
KR2	KR4	72.78	41.31
KR3	KR1	72.45	35.62
KR3	KR2	72.44	35.69
KR3	KR4	72.44	35.82
KR4	KR1	72.67	42.81
KR4	KR2	72.67	43.06
KR4	KR3	72.67	43.21

Table 5.50: RTT values in Group G4-2 with LBS algorithm

Destination	Origin	RTT AVG	RTT STD
KR1	KR2	89.36	29.17
KR1	KR3	3.93	8.89
KR1	KR4	64.79	33.24
KR2	KR1	89.23	28.81
KR2	KR3	99.70	31.79
KR2	KR4	139.71	47.43
KR3	KR1	4.25	8.17
KR3	KR2	101.93	30.77
KR3	KR4	74.50	31.56
KR4	KR1	89.66	38.79
KR4	KR2	135.76	42.81
KR4	KR3	96.54	35.62

Group G4-3: (Pre-measured One-Trip Time)

The OTT values between nodes in group G4-3 with LBS algorithm is shown in Table 5.51. As it is explained in previous sections, these values are measured before each

session begins. Therefore, the values can be different from the half of RTT values measured during game sessions due to unstable network environment. The smaller the pre-measured OTT values is than the half of RTT values which are assessed during game sessions, the larger the packet resending rate is. On the opposite case, frame interval can be prolonged due to delayed packet resending. More detail analysis will be provided in following analysis section.

Table 5.51: One-Trip Time between nodes in G4-3 with LBS

Node No.	AU1	AU2	KR1	KR3
AU1		5.85	165.25	174.75
AU2	5.85		169.58	165.75
KR1	165.25	169.58		13.65
KR3	174.75	165.75	13.65	

Group G4-3: (Overall Performance)

The overall performance such as FPS, drop rate, resend rate, and PPS values are shown in Table 5.52. The FPS value of nodes in group G4-3 with LBS algorithm is approximately 18.9. This value is close to the FPS value of nodes in group G4-3 with FSR algorithm and it denotes about 400% of performance improvement compared to the experimental results from the same group with LS algorithm in terms of game execution speed. Further analysis will be presented in following analysis section.

Table 5.52: FPS, PPS, Drop Rate in G4-3 with LBS

Node	FPS:	Drop Rate (IN)	Resend Rate (OUT)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	18.99	0.00%	0.39%	114.14	114.35	228.49
AU2	18.99	0.00%	0.39%	114.09	114.31	228.40
KR1	18.92	0.00%	0.52%	113.92	114.13	228.05
KR3	18.93	0.00%	0.53%	113.97	114.20	228.17

Group G4-3: (Frame Interval & RTT)

The lower packet resending rates of nodes in group G4-3 with LBS algorithm are originated from the regularity of RTT values which is shown in Table 5.54. The RTT

standard deviation values indicate the regularity of RTT values which are measured during game sessions, and the divergence rate of these values is less than 10% of the average RTT values.

By recalling the rules for packet resending time which is 110% of pre-measured OTT value between nodes, the packet resending rate can be minimized if the standard deviation value of RTT is less than the delay value, which will be added to the pre-measured OTT value for packet resending between nodes. Again, further analysis will continue in the following analysis section. Table 5.53 and Table 5.54 present the frame interval and RTT values between nodes in group G4-3 with LBS algorithm.

Table 5.53: Frame interval in Group G4-3 with LBS algorithm

Destination	Origin	FI AVG	FI STD
AU1	AU2	52.64	21.58
AU1	KR1	52.64	21.57
AU1	KR2	52.64	21.58
AU2	AU1	52.68	16.04
AU2	KR1	52.68	16.03
AU2	KR2	52.68	16.03
KR1	AU1	52.82	16.27
KR1	AU2	52.81	16.28
KR1	KR2	52.81	16.28
KR2	AU1	52.82	17.97
KR2	AU2	52.82	18.23
KR2	KR1	52.82	18.33

Table 5.54: RTT values in Group G4-3 with LBS algorithm

Destination	Origin	RTT AVG	RTT STD
AU1	AU2	5.25	2.26
AU1	KR1	327.98	22.93
AU1	KR2	345.60	27.21
AU2	AU1	3.79	2.63
AU2	KR1	337.12	25.73
AU2	KR2	330.34	26.79
KR1	AU1	326.43	23.87
KR1	AU2	327.44	25.44
KR1	KR2	3.73	8.49
KR2	AU1	343.38	27.94
KR2	AU2	328.02	25.68
KR2	KR1	7.17	9.35

Group G8-1: (Pre-measured One-Trip Time)

The pre-measured OTT values between nodes in group G8-1 with LBS algorithms are presented in Table 5.55. As it can be seen in the table, the maximum OTT value is 238.73 ms between node AU4 and KR2. The connection between node KR2 and other nodes seems very bad in terms of network latency compared to other nodes' connection. Probably it is caused by its network environment and heavy usage of the network.

Currently the node KR2 adopts wireless network and total three computers share one link to the Internet. During the experimental sessions, other two users were downloading large files and watching multimedia contents. It is assumed that these kinds of activities generate the bad system performance but it has not been verified and validated yet due to the irrelevance from the research topic and area.

Table 5.55: One-Trip Time between nodes in G8-1 with LBS algorithm

Node No.	AU1	AU2	AU3	AU4	KR1	KR2	KR3	KR4
AU1		9.23	5.33	6.50	164.73	231.60	179.95	189.58
AU2	9.23		7.50	8.98	169.15	236.00	174.50	182.23
AU3	5.33	7.50		4.38	176.38	226.70	186.05	196.53
AU4	6.50	8.98	4.38		164.25	238.73	176.68	186.00
KR1	164.73	169.15	176.38	164.25		80.70	9.78	33.38
KR2	231.60	236.00	226.70	238.73	80.70		84.03	93.85
KR3	179.95	174.50	186.05	176.68	9.78	84.03		39.83
KR4	189.58	182.23	196.53	186.00	33.38	93.85	39.83	

Group G8-1: (Overall Performance)

FPS, PPS, drop rate, and packet resending rate are measured during game sessions and calculated afterward. These values of group G8-1 are presented in Table 5.56. The FPS values are around 12.5 which is about 360% improvement compared to the FPS values of group G8-1 with LS algorithm. It shows that the playout delay of LBS algorithm can mask network latency and improve game execution speed. More details

will be given in following analysis section. As mentioned before, the packet resending rate of node KR2 is considerably larger than other nodes', and it may cause more outbound packets to other nodes.

Table 5.56: FPS, PPS, Drop Rate in G8-1 with LBS algorithm

Node	FPS:	Drop Rate (IN)	Resend Rate (OUT)	PPS (IN)	PPS (OUT)	PPS (ALL)
AU1	12.54	0.00%	3.29%	181.19	181.59	362.78
AU2	12.53	0.00%	3.81%	182.15	182.53	364.68
AU3	12.54	0.00%	2.77%	180.27	180.64	360.91
AU4	12.54	0.00%	2.90%	180.51	180.84	361.35
KR1	12.53	0.00%	7.84%	189.81	190.19	380.00
KR2	12.44	0.02%	13.61%	201.01	201.62	402.63
KR3	12.53	0.00%	7.49%	189.09	189.49	378.58
KR4	12.53	0.00%	7.38%	188.63	189.05	377.68

Group G8-1: (Frame Interval)

Most of standard deviation RTT values of node KR2 are 20%larger than that of the overage RTT values. This implies that node KR resends packets frequently due to delayed acknowledgement packets. Table 5.57 and Table 5.58 show the frame interval and RTT values which are measured during game sessions, separately.

5.4. Comparison and Analysis

Comparison of FPS and PPS values between FSR, LS and LBS is presented in this section. Also, the analyses for divergence in FRS, frame interval and, RTT are also presented here.

Table 5.57: Frame interval in Group G8-1 with LBS

Destination	Origin	FI AVG	FI STD
AU1	AU2	77.93	60.00
AU1	AU3	77.93	59.97
AU1	AU4	77.93	59.99
AU1	KR1	77.93	60.20
AU1	KR2	77.93	60.25
AU1	KR3	77.93	60.34
AU1	KR4	77.93	60.43
AU2	AU1	77.92	54.40
AU2	AU3	77.92	54.35
AU2	AU4	77.92	54.41
AU2	KR1	77.92	54.35
AU2	KR2	77.92	54.38
AU2	KR3	77.92	54.37
AU2	KR4	77.92	54.39
AU3	AU1	77.94	55.13
AU3	AU2	77.94	55.36
AU3	AU4	77.94	55.60
AU3	KR1	77.94	55.92
AU3	KR2	77.94	56.15
AU3	KR3	77.94	56.30
AU3	KR4	77.94	56.53
AU4	AU1	77.90	55.18
AU4	AU2	77.90	55.18
AU4	AU3	77.90	55.18
AU4	KR1	77.90	55.19
AU4	KR2	77.90	55.18
AU4	KR3	77.90	55.19
AU4	KR4	77.90	55.22
KR1	AU1	77.96	47.96
KR1	AU2	77.96	47.96
KR1	AU3	77.96	47.96
KR1	AU4	77.96	47.96
KR1	KR2	77.96	47.96
KR1	KR3	77.96	47.96
KR1	KR4	77.96	47.96
KR2	AU1	78.58	60.27
KR2	AU2	78.58	60.72
KR2	AU3	78.58	60.85
KR2	AU4	78.58	61.09
KR2	KR1	78.58	61.28
KR2	KR3	78.58	61.48
KR2	KR4	78.58	61.68
KR3	AU1	77.96	48.20
KR3	AU2	77.96	48.16
KR3	AU3	77.96	48.14
KR3	AU4	77.96	48.25
KR3	KR1	77.96	48.19
KR3	KR2	77.96	48.20
KR3	KR4	77.96	48.01
KR4	AU1	78.23	48.55
KR4	AU2	78.23	48.57
KR4	AU3	78.23	48.61
KR4	AU4	78.23	48.69
KR4	KR1	78.23	48.74
KR4	KR2	78.23	48.78
KR4	KR3	78.23	48.81

Table 5.58: RTT values in Group G8-1 with LBS

Destination	Origin	RTT AVG	RTT STD
AU1	AU2	13.23	4.67
AU1	AU3	14.63	8.70
AU1	AU4	11.32	4.72
AU1	KR1	323.69	24.02
AU1	KR2	470.39	79.31
AU1	KR3	361.66	28.16
AU1	KR4	380.52	42.15
AU2	AU1	7.76	5.55
AU2	AU3	7.98	8.76
AU2	AU4	5.41	2.61
AU2	KR1	335.92	24.45
AU2	KR2	477.61	81.19
AU2	KR3	344.83	26.72
AU2	KR4	363.57	40.88
AU3	AU1	8.82	9.64
AU3	AU2	8.10	8.97
AU3	AU4	8.74	9.03
AU3	KR1	341.38	26.46
AU3	KR2	469.78	78.60
AU3	KR3	366.30	29.48
AU3	KR4	384.84	45.55
AU4	AU1	2.27	6.05
AU4	AU2	0.75	3.35
AU4	AU3	4.33	10.52
AU4	KR1	326.47	23.62
AU4	KR2	476.55	81.08
AU4	KR3	344.74	27.38
AU4	KR4	366.26	39.53
KR1	AU1	314.07	26.34
KR1	AU2	326.31	26.49
KR1	AU3	330.19	28.50
KR1	AU4	326.02	26.38
KR1	KR2	167.62	69.84
KR1	KR3	21.27	10.82
KR1	KR4	64.43	28.49
KR2	AU1	470.57	83.26
KR2	AU2	481.12	84.53
KR2	AU3	473.52	82.51
KR2	AU4	484.43	84.35
KR2	KR1	178.49	65.24
KR2	KR3	196.71	67.22
KR2	KR4	206.95	70.27
KR3	AU1	361.16	33.66
KR3	AU2	344.35	29.12
KR3	AU3	365.32	32.14
KR3	AU4	344.76	29.24
KR3	KR1	27.20	8.98
KR3	KR2	190.14	68.69
KR3	KR4	92.76	30.10
KR4	AU1	385.11	39.80
KR4	AU2	369.21	40.40
KR4	AU3	389.43	41.73
KR4	AU4	373.24	40.55
KR4	KR1	86.83	30.97
KR4	KR2	211.57	74.80
KR4	KR3	102.49	30.32

5.4.1. FPS and PPS comparison

Four groups, G4-1, G4-2, G4-3, and G8-1, are used with three consistency maintenance algorithms, FSR, LS, and LBS for comparison of FPS and PPS. To provide a clearer and simpler comparison, node AU1 is chosen in group G4-1, G4-3, G8-1 and node KR1 in group G4-2. As explained in previous sections, group G4-1 consists of four nodes in the LAN of Griffith University in Australia. Another four-node-group, G4-2, resides in South Korea, and usual latency between nodes in this group is less than 80 ms. The third four-node-group, G4-3, consists of two nodes in Australia and two nodes in South Korea. To minimize the effect of slow game execution time and latency in domestic network (AU1-AU2 and KR1-KR3 links) on the experimental results in the game sessions, node AU1, AU2, KR1, and KR3 are selected. Finally, group G8-1 includes all eight nodes which are used in each group.

The measured and calculated FPS values in each group are denoted in Figure 5.21. As it can be seen, node AU1 with FSR algorithm achieves upper benchmark speed which is about 25 fps and node KR1 shows 21.33 fps. These values are upper benchmark performance for each node and compared to others with LS and LBS algorithms.

Among three algorithms, LS algorithm has worst performance in terms of game execution speed. It achieves about 20 fps in group G4-1 and this value is also almost upper benchmark value because the slowest game execution speed in the group is about 21 fps. LS algorithm's "send-and-wait" scheme synchronises node's game execution speed with the one of the slowest node's. However, the performance of this algorithm drops significantly in group G4-2, G4-3, and G8-1 which are 7.28, 4.80, and 3.45, respectively. The game execution speed in these groups is recognizably slow and it is frustrating to control ships when auto-pilot function is turned off.

LBS algorithm also achieves upper benchmark speed in group G4-1 and the FPS value goes down to 13.8 in group G4-2 mainly due to the system performance of the slowest node in the group and unstable packet delay between nodes. Node KR4's game execution speed is 14.32 fps with FSR algorithm and packet resending rate is 15.45% with LBS algorithm in this group.

The FPS value goes up again in group G4-3 and this value is not upper benchmark value. The slowest game execution speed in this group is 21.31 from node KR3 and measured packet resending rates of nodes are also very small, they are generally from 0.39% ~ 0.53%. The most significant factor that affects game execution speed is network latency between AU nodes and KR nodes which is 174.75ms, a maximum average OTT value, between node AU1 and KR3. This value is slightly larger than pre-defined playout delay and it causes the slowdown of game execution speed.

Finally, the FPS value goes down again in group G8-1 due to network latency in international links and the game execution speed of the slowest node in this group. The FPS value is 12.54 and it is about 50% of full game execution speed. However, the game execution speed is not noticeably slow compared to the game sessions in the same group with LS algorithm. Also, it is quite controllable when manual pilot is on.

The number of packets sent per second in game sessions are also measured and depicted in Figure 5.22. Again, node AU1 and KR1 are selected for further comparison and analysis. In the figure, group G4-1 shows general comparison graph of PPS values with FSR, LS, and LBS. As it is explained in previous sections, FSR algorithm does not transfer acknowledgement packets but game packets only on regular interval.

Therefore, if the regular interval is identical between FSR, LS, and LBS algorithms, LS and LBS algorithms transmit double number of packets compared to FSR algorithm. PPS value increases when FPS value increases and it is well depicted in Figure 5.22. Therefore, LS and LBS algorithms may suffer from overwhelmed number of packets compared to FSR algorithm. However, the actual size of packet in LS and LBS can be much smaller than the packet size in FSR algorithm.

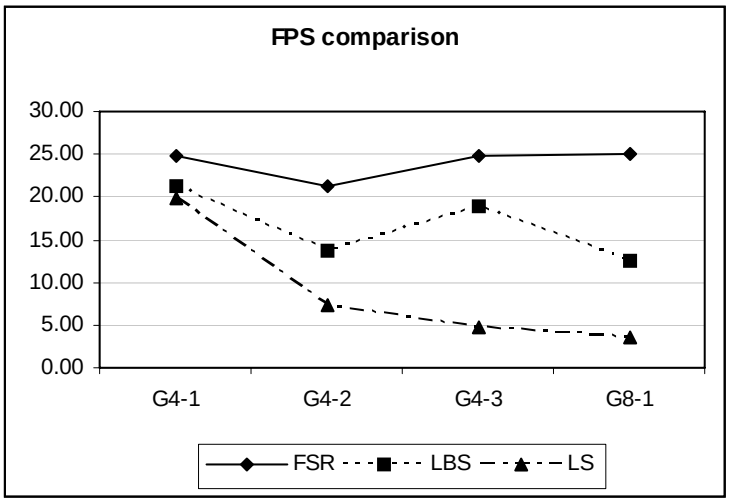


Figure 5.21: FPS comparison between FSR, LS, and LBS

As it is mentioned before, FSR algorithm sends the state of objects in game sessions but LS and LBS algorithm can transmit state of object and/or command from players. Usually, sending command requires much smaller number of bytes than transmitting the state of objects. Also, FSR requires sending full size game packets which contain the state information of objects whether these are identical to previously sent packets or not. On the other hand, LS and LBS algorithm can send empty packet if command is transmitted and there is no command to send. Therefore, it is not always true that LS and LBS algorithms require more bandwidth requirement than FSR algorithm.

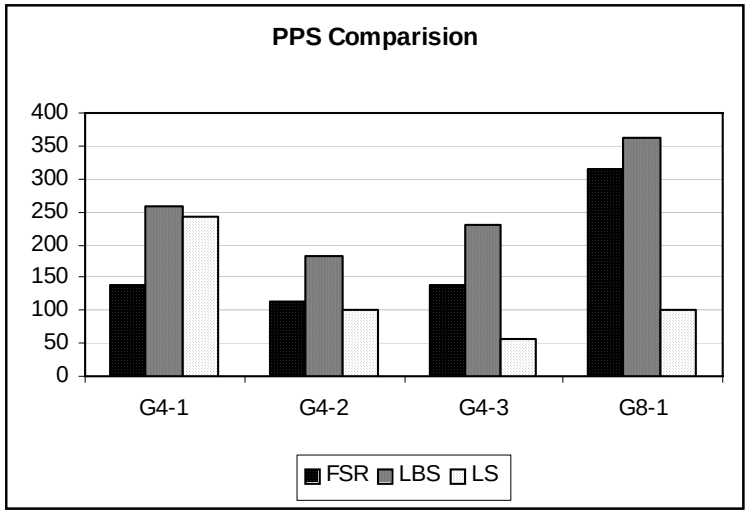


Figure 5.22: PPS comparison between FSR, LS, and LBS

5.4.2. Divergence (FSR only)

The divergence between real and ghost objects in nodes is measured and calculated during game sessions of each group with FSR algorithm. Tables and figures that related to the divergence are presented in each section of all four groups and various factors that cause the divergence are explained.

These three factors that affect the divergence between real and ghost objects are:

- Network latency
- Game execution speed
- Game starting time

The first factor, network latency, is explained in group G4-1. In this case, the average divergence values between node AU2 and AU4 are 0.67 and 0.49 for divergence X and Y, respectively.

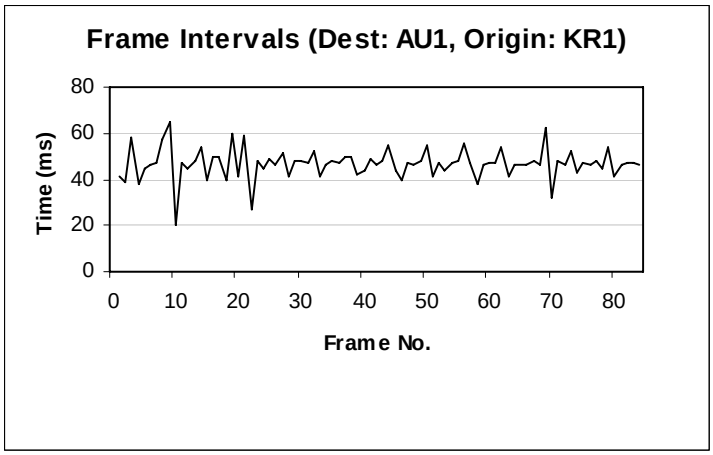
The effect of unsynchronised game execution speed and game starting time on divergence is fully explained in each section for all groups with FSR algorithm. As it is mentioned in previous sections, unsynchronised game execution speed distorts the state of objects in frame time line and it can harm consistency of game state between players. Also, the fairness of game playability is hard to maintain under this situation if it is not impossible. Unsynchronised game starting time may reduce the divergence between real and ghost object in one node but it causes opposite effect on the other node as it is covered in group G4-3 and G8-1. Therefore, time synchronisation mechanism is required when multi-player games adopt FSR algorithm.

5.4.3. Frame Interval and RTT

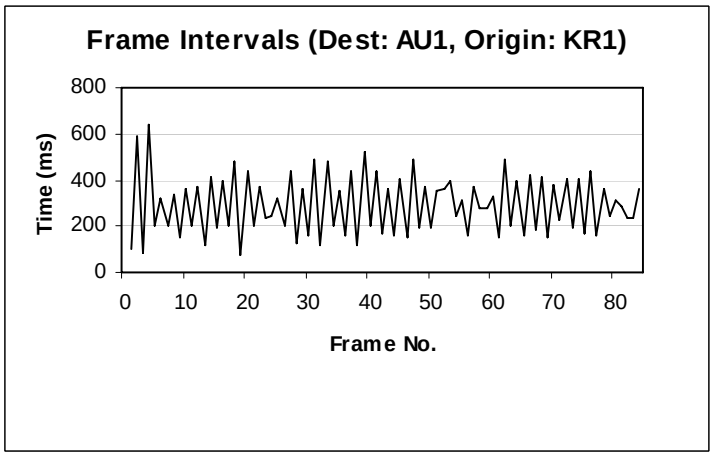
Frame interval and RTT values between nodes in each group are measured during game sessions. The section of each group covers the characteristics of each algorithm in terms of frame interval. Therefore, in this section, comparison between algorithms will be provided. Figure 5.23 shows the comparison in which the average frame interval value of node KR1 in group G8-1 with FSR is 46.87 and the standard deviation value is 5.16. The second one depicts frame interval values for each frame with LS algorithm and the average value is 288.57 and the standard deviation value is 119.54. Figure 5.23 (c) shows ones with LBS algorithm and average and standard deviation values are 77.95 and 60.20, respectively.

FSR algorithm performs best in terms of the regularity of frame interval and it can be denoted by the standard deviation value which is 5.16. LS algorithm shows large fluctuation graph and its STD value is 119.54 due to network latency between node AU1 and KR1. This implies that the frame interval is irregular, therefore, the game playability is degraded in terms of smoothness of game play. Finally, the graph of

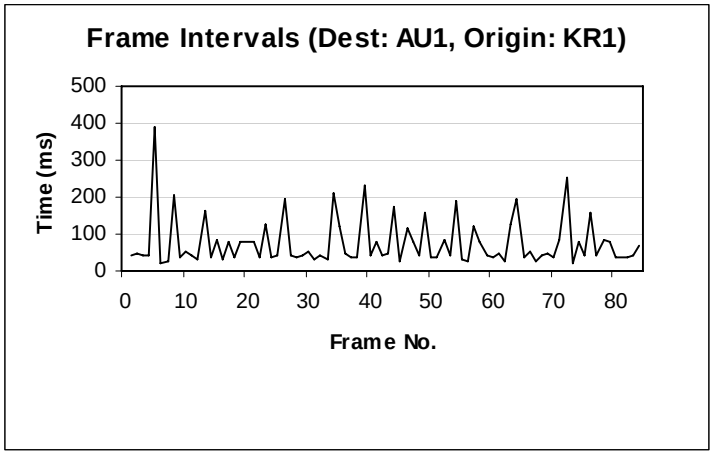
frame interval values with LBS algorithm is presented in Figure 5.23 (c). Clearly, the graph shows the abilities of LBS algorithm which are masking network latency and reducing the fluctuation of frame interval.



(a) Node KR1's frame interval with FSR algorithm



(b) Node KR1's frame interval with LS algorithm



(c) Node KR1's frame interval with LBS algorithm

Figure 5.23: Comparison of Frame Interval with FSR, LS, and LBS

The comparison between pre-measured OTT and in-game-measured RTT values is provided in the section of group G8-1 with LS algorithm. The divergence between two values may cause high packet resending rate if pre-measured OTT value is under estimated. In opposite case, game execution speed can drop due to delayed packet resending. Also, additional delay value for setting up packet resending time is important to achieve upper benchmark performance in terms of packet resending rate and game execution speed. When the fluctuation of RTT value is larger than the additional delay value then packet resending occurs. Therefore, this suggests that careful observation on network latency and dynamic decision on the additional delay and OTT time based on the change of network situation may improve the game performance.

5.5. Chapter Summary

In this chapter, we proposed an algorithm, called Locked Bucket Synchronization (LBS), to maintain consistency in P2P multiplayer distributed network games. The

proposed approach makes use of human perception delay to mask network delay. For comparison, two other algorithms are also considered, namely Frequent State Regeneration (FSR) and Lockstep (LS), both of which are well known in the existing literature. FSR is essentially optimistic, in that it immediately commits local and remote commands to processing. By contrast, both LS and LBS are conservative, in that they go ahead with processing only if the commands are consistent.

In practice, FSR transmits packets of data at regular intervals, thus necessitating the transmission of significant amounts of data. The larger the numbers of participants and objects in MODIGs, the more serious this issue becomes. Further, FSR has a limited capacity to resolve divergences and inconsistencies. By contrast, LS and LBS wait and disseminate packets only when the safety of process execution can be guaranteed. The major difference between LS and LBS algorithms is that the latter has a buffering functionality that can absorb the impact of network latency and jitter.

The relative efficiency of the three algorithms is tested within the context of two applications, COMP2P and Duel-X. The experiments using COMP2P involve the testing of other latency-alleviating approaches (addressing packet transmission and network structure) and will be discussed in Chapter 6. Experiments with Duel-X are reported in the main body of this chapter.

The Duel-X experimental results confirm that, as can be expected, FSR performs very well in terms of game execution speed, ranging from 16.16 to 24.93 fps (frames per second) compared with the maximum possible speed of 25 fps. However, due to unsynchronised game execution speed and game starting time across player nodes, it performs very poorly in terms of maintaining consistency of the game states of the

various players. Consequently, players are likely to experience inconsistencies and confusion, which may result in a sense of unfairness and irritation.

By contrast, the LS algorithm maintains perfect consistency, but suffers from slow game execution speed because of its "send-and-wait" philosophy. Its speed ranged from 3.45 to 19.88 fps -- in many instances, about one-third of the speed provided by the FSR algorithm. This means that game sessions played with LS are likely to result in slow game execution and considerable player frustration.

The results also suggest that LBS offers an attractive alternative which avoids the worst features of the above two polar approaches, while retaining their essential strengths. On one hand, LBS maintains perfect consistency, thus beating FSR and matching LS as far as consistency is concerned. On the other, it consistently outperforms LS in terms of execution speed. Even in the worst cases, LBS performs at approximately 12.5 fps, or about 50% of the maximum possible speed, which is quite playable.

It is interesting to note that the execution speed of the LBS algorithm reaches 21.22 fps, or about 85% of the maximum possible speed, when network latency is shorter than playout delay, which is the time elapsed between a particular user's initiation of a command and the appearance of the result. In that case, the key operative constraint on the system's execution speed is the *slowest* node's game execution speed, and this is unrelated to the design of the network environment.

6. Latency Alleviation and Bandwidth Requirement Reduction

6.1. Introduction

In this chapter, a number of algorithms and approaches are proposed with the aim of reducing the extent and impact of latency, as well as bandwidth requirements. These approaches include a smart transmission scheme, a tree-based P2P network structure, and a packet aggregation method. Before turning to a detailed description of each proposed innovation, as well as its implementation and experimental results, it may be useful to present a brief overview of the rationale for introducing all three innovations.

When packets drop, game execution speed can deteriorate considerably, and the extent of the deterioration depends on the significance of the dropped packets. In the Frequent State Regeneration (FSR) algorithm, none of the packets are truly significant because of offsetting characteristics of this algorithm, such as relatively short periods of inconsistency, LAN-based systems, and a simple consistency maintenance mechanism. FSR can achieve relatively short periods of inconsistency between players' states (in comparison with, for example, Dead Reckoning, Time Warp, LS, and LBS) through frequent periodic transmissions. Further, FSR is typically implemented in a LAN environment due to its significant bandwidth requirements. In any case, inconsistency is simply not a priority in FSR.

By contrast, LS and LBS pursue the global ordering of operations requested by users through the assurance that operation commitment is safe, in that the operations to be committed are consistent with one another. Because of this strict rule, dropped

packets can cause significant delays, especially when the dissemination delay of packets between links is longer than frame interval. If n is this packet dissemination delay, then when a packet is dropped, the time required to receive the retransmitted packet is at least $3n$.

To minimize such retransmission delays and to reduce the overall latency of the system, the Smart Transmission Scheme (STS) is proposed. STS checks the future arrival times of packets to determine whether or not any given packet that is being sent is critical, in the sense that if it is dropped the game will be delayed. If a packet is critically important, it is sent twice. Further details about STS as well as the related experimental results on COMP2P are presented in Section 6.2.

Another way to alleviate latency is to use a network structure that shortens packet propagation delays. Most existing P2P games use a graph structure to form a network topology, where each node connects to all other nodes. This practice can cause bandwidth problems if unicasting is used. Furthermore, since all nodes are synchronised with the slowest node, this structure cannot take advantage of fast connections between any subsets of nodes.

As an alternative, if network topology is structured by a tree-based architecture then nodes will be hierarchical and fast connection nodes can be grouped together. In the current context the most important implication of this is that a tree structure can increase overall game speed by reducing the network latency of the slowest link in the network. In Section 6.3, we discuss graph to tree conversion, as well as relay, retransmission, and acknowledgement mechanisms. Further, experimental results on COMP2P with both the LS and LBS algorithms are reported and analysed.

While the tree structure can reduce overall network latency and the bandwidth requirements of some child nodes, it implies significant bandwidth requirements for parent nodes whose child nodes do not possess direct connections with each other. Given an identical amount of data to be transferred, the bandwidth requirement in a tree-based P2P network can differ quite significantly from that of a graph-based network. In the latter network, the bandwidth requirement of each player is a linear function of the number of players. In a tree-based system, by contrast, it is a quadratic function of the number of child nodes. This implies that, due to limited bandwidth, some nodes (acting as parent nodes) may at times be overwhelmed by the arrival of a large number of packets, leading to a packet-drop. In turn, the packet-drop would trigger packet retransmissions. This may result in a repetitive cycle of packet retransmissions and packet-drops, possibly resulting in severe network latency.

To minimize these negative effects, a packet aggregation approach is proposed in Section 6.4, where further information about its implementation and experimental results is also provided. Section 6.5 presents a summary of the main findings of the chapter.

6.2. Packet Transmission Schemes

In this section, a so-called smart transmission scheme is proposed to reduce delays associated with the retransmission of dropped packets. For comparison, a blind transmission scheme is also considered. (Some of the results reported below have been presented in a work-in-progress format in Moon et al., 2005.)

A major advantage of the Blind Transmission Scheme (BTS) is the simplicity of its implementation. As the name implies, under this scheme any given packet of

information would be sent twice. While this scheme does reduce the delays associated with dropped packets, it suffers from a disadvantage in that its bandwidth requirement for each player will be nearly double that of the normal transmission scheme (i.e., no special transmission scheme - NTS). Under the latter scheme, update packets are transmitted only once. A packet will be retransmitted only if it is significant *and* packet-drop or corruption is detected (these conditions do not apply if the FSR algorithm is adopted: then packets are sent at regular intervals).

Lincroft (1999) used an aggregated packets transmission scheme in his "X-Wing vs. TIE Fighter" game whereby the previous frame and the current frame packets are aggregated together before being transmitted. This scheme also doubles bandwidth requirements, similar to the requirements for BTS. Doubling bandwidth requirements is usually not a favoured feature in designing multiplayer games because it will constrain game design significantly.

The aim of the smart transmission scheme (STS) is to reduce latency, just as the BTS does, but without increasing bandwidth requirements to the same extent. It does this by checking the future arrival time of a given packet to determine whether or not the packet is critical to the execution of the game. This is achieved by comparing two values, the must-arrive time and the next packet arrival time. The must-arrive time (MAT) is the time when the packet must arrive so as not to delay the game process. It depends on the playout time of the game. In the Lockstep algorithm, the MAT is equal simply to the playout delay: packets must arrive before the next frame starts. In the LBS algorithm, the MAT is the sum of packet sending time and the playout delay. The next packet arrival time (NPAT) is the time when the next packet arrives if

a packet drop is detected and the packet is retransmitted. If the packet being sent is determined to be a critical packet then it is sent twice, as a precaution.

6.2.1. Experimental Results

Two consistency maintenance algorithms, Lockstep and Locked Bucket-Synchronization algorithm (LBS), are implemented with two transmission schemes, blind and smart (BTS and STS) for this experiment. The experiment duration is 60 seconds and frame interval is 100 ms because the maximum network latency is set as 100 ms between players. Playout delay for LBS is 200 ms and three player groups are used which are 4, 8 and 16 player groups.

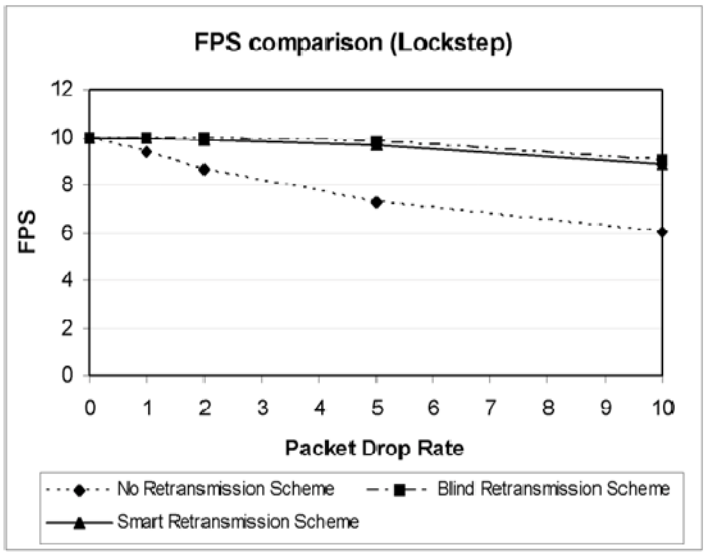
Table 6.1, Figure 6.1 and Figure 6.2 show experimental results which are based on Lockstep algorithm and two types of transmission schemes. The first row in the table shows the optimal results of the game session which has no packet drop at all. Therefore, FPS (Frame per Second) value is 10 and this value is same as the optimal value. In optimal situation, the formula for the total number of packets per frame is expressed as $n(n-1)\times 2$ where n is the number of players. A factor of 2 is included because we also count acknowledgement packets even though their size is smaller than the size of frame packets. Therefore, the value for the Average Packets per Frame column in Table 4 should be 24. However, according to the Table, the value is 23.95 because the experiment only counted packets which arrived in 60 seconds sharp.

Table 6.1: Four player game session: Experimental results of Lockstep algorithm and No Transmission Scheme

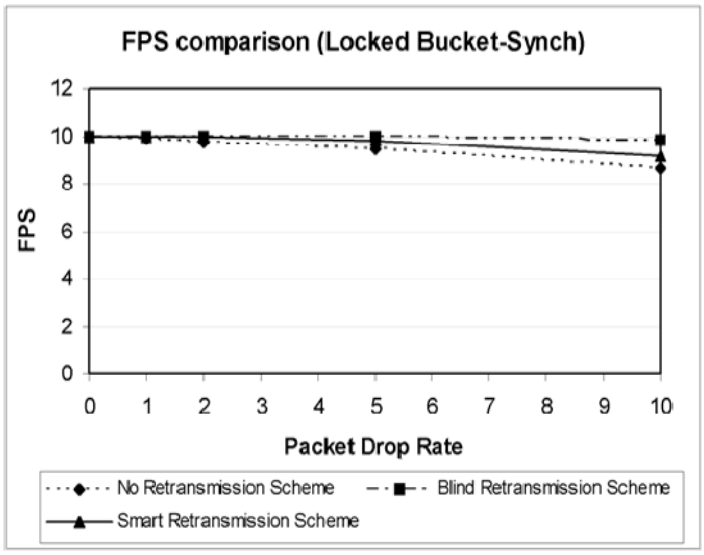
Drop Rate (%)	Total Packets	Dropped Packets	FPS	Avg. Packets per Frame
0	14368	0	10.00	23.95
1	13679	117	9.40	24.25
2	12755	233	8.63	24.62
5	11317	597	7.25	26.02
10	10134	998	6.05	27.92

When the Blind Transmission Scheme (BTS) is applied, the number of packets transmitted doubled. It can be observed clearly in Figure 6.2. However, the FPS values are increased positively. With 10% of packet drop rate in lockstep and no transmission scheme combination in Table 6.1, the FPS dropped down to 6.05 which is 60% of full game execution speed. However, with BTS and 10% packet drop rate, the FPS became 9.02 in Figure 6.1. Furthermore, Figure 6.1 and Figure 6.2 show that the BTS performs well without variation compare to the no special transmission scheme (NTS) with lockstep algorithm.

Also the experimental result with Lockstep algorithm and Smart Transmission Scheme (STS) is shown in Figure 6.1 and Figure 6.2. According the result, STS reduces the total number of packets but the FPS values are almost identical to those of BTS. However, when the packet drop rate increases, the total number of packets per frame increases and becomes very similar to the corresponding one in BTS. For example, STS with Lockstep sends about 40 packets per frame when the packet drop rate is 10%, which is not so different from BTS with Lockstep that is 46 packets per frame.



(a)



(b)

Figure 6.1: Four-player-game sessions: The changes of FPS according to packet drop rate on Lockstep and LBS algorithms with Transmission schemes

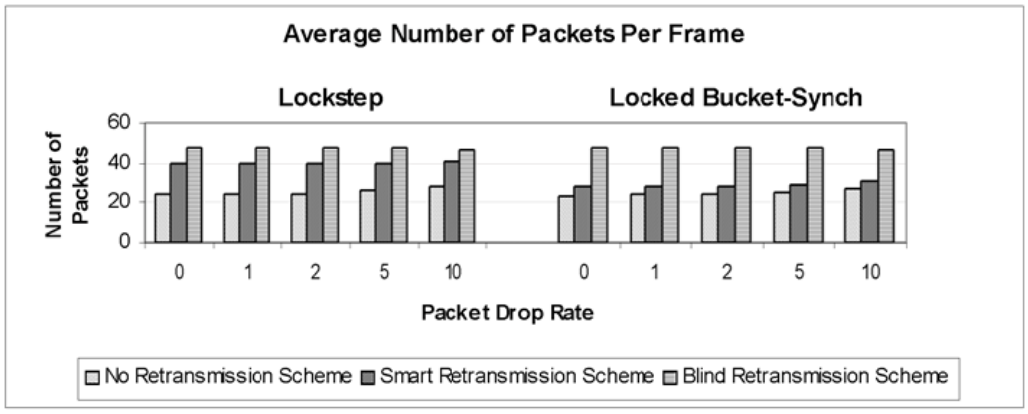
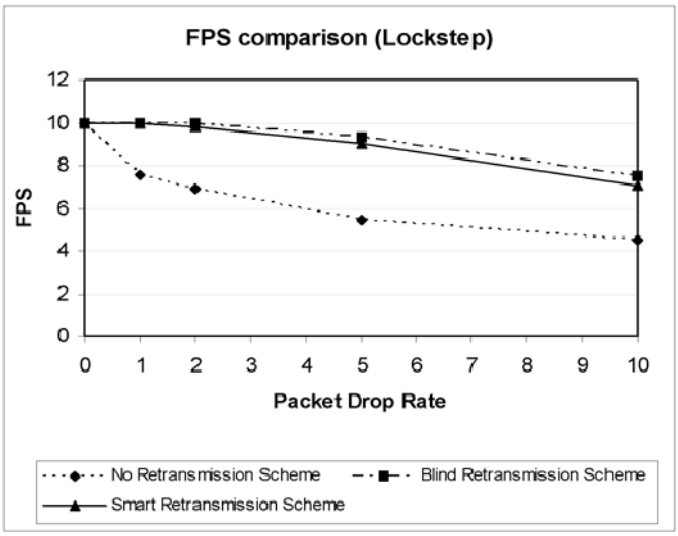


Figure 6.2: Four-player-game sessions: The changes of average number of packets per frame according to packet drop rate on Lockstep and LBS algorithms with Transmission schemes

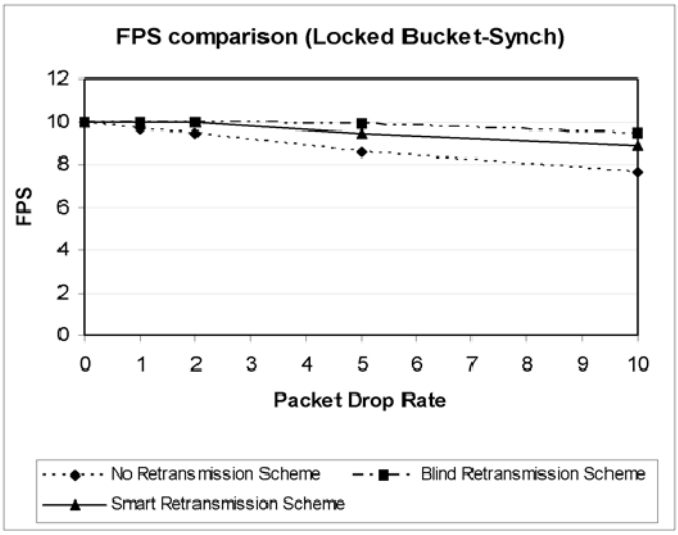
Figure 6.3 and Figure 6.4 shows the changes of FPS in eight and sixteen-player-game sessions. The graphs of the changes of bandwidth requirement are omitted due to similarity of their shapes compared to Figure 6.2. As can be observed, the more players and higher drop rate, the poorer the performance (FPS) is without transmission schemes.

This situation gets worse when the Lockstep algorithm is used. In Lockstep and transmission schemes with eight player game, about 75% of full game execution speed is achieved, but only 50% in 16 player game sessions with Lockstep and any transmission schemes combinations. This clearly shows that not only bandwidth requirement is critical in supporting more players in network games, but also network latency is when conservative algorithms are used for consistency maintenance. To mask network latency, we propose the LBS algorithm.

Figure 6.4 shows that it performs better than the Lockstep algorithm, in which it executes game sessions at about 80% of the full game speed even with 10% packet drop rates in sixteen-player game sessions when transmission schemes are applied.



(a)



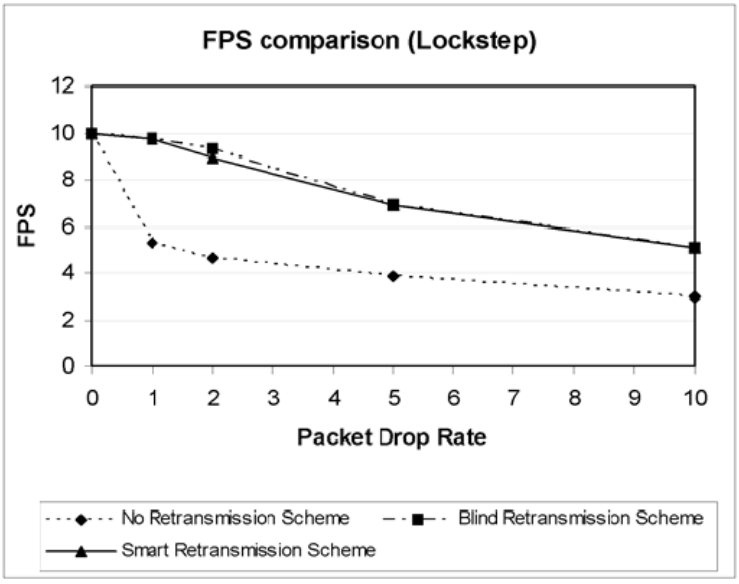
(b)

Figure 6.3: Eight-player-game sessions: The changes of FPS according to packet drop rate on Lockstep and LBS algorithms

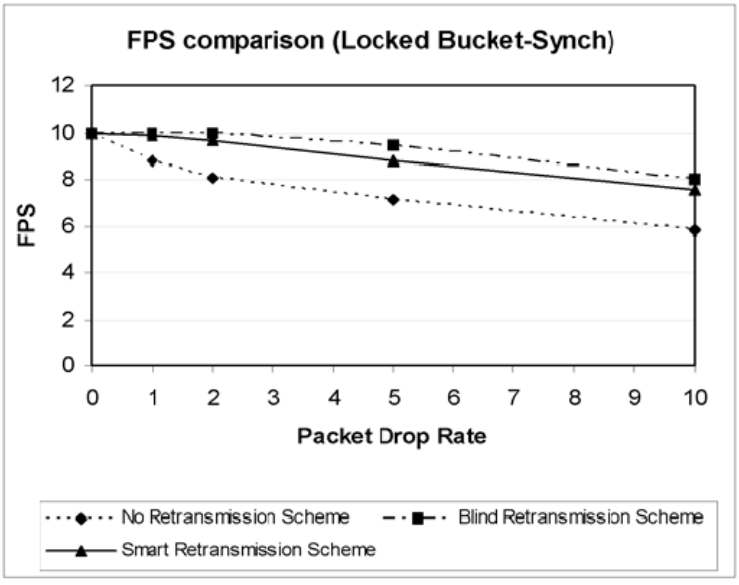
6.2.2. Analysis

Figure 6.1 and Figure 6.2 show the changes of FPS and the average number of packets per frame in four-player-game sessions. As can be seen, the proposed Locked

Bucket-Synchronization (LBS) algorithm performs better than the Lockstep algorithm in terms of game execution speed (FPS) and bandwidth requirement.



(a)



(b)

Figure 6.4: Sixteen-player-game sessions: The change of FPS according to packet drop rate on Lockstep and LBS algorithms with Transmission schemes

With BTS and LBS combination, the total number of packets is almost identical to the BTS and lockstep algorithm combination. With STS and LBS, the total number of packets becomes only about 10% more than the LBS without transmission scheme, but the FPS value is almost optimal which is above 92% of full game speed in any drop rate case.

Furthermore, Figure 6.1 (a) clearly shows the effect of packet drops compared to Figure 6.1 (b) which utilized LBS algorithm. According to the figures, FPS values drastically decreased as the packet drop rates goes up with lockstep algorithms without transmission scheme. However, LBS algorithm performs well even without transmission schemes.

6.3. Tree-based P2P System

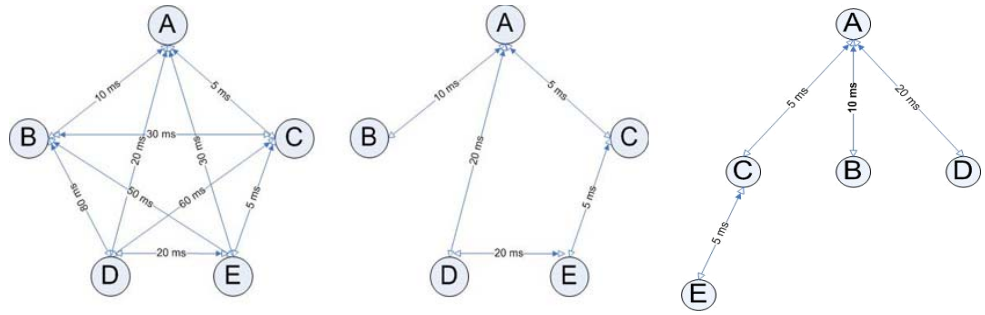
To maintain fairness for all players in a game session, the game speed is synchronized with the slowest node in terms of process power and/or network latency. Thus there are two variables to be considered: process speed and network latency, both of which vary from node to node. Of these, the first variable -- process speed -- is easy to test before the game session starts, and it usually does not change dramatically during the game. Therefore, the process speed question can be resolved by pre-testing whether a player's computer can process an adequate frame rate during game sessions. By contrast, network latency is harder to predict; moreover, it tends to vary during game sessions. In this section, we propose a tree-based P2P system to address some aspects of this problem.

6.3.1. Graph-based and Tree-based Distributed Network Games

Distributed MODIGs (Multiplayer Online Digital Games) can be organized by using graph or tree structures. Generally, a complete graph structure is preferable to a tree structure due to the simplicity of implementation. It is assumed that the graph structure is a connected graph throughout this chapter unless otherwise stated. The tree structure is a converted form of graph structure by arranging each node in the view of a root node. The root node is a player and it establishes links (in other terms, edges or arcs) to other nodes directly or indirectly depending on network variables such as latency and/or bandwidth.

Existing P2P games use a graph structure to form a network topology, where each node connects to all other nodes. As explained before, it can cause bandwidth problem if unicasting is used. Furthermore, since all nodes are synchronized with the slowest node, this structure can not take advantage of fast connections between any other nodes.

By contrast, if network topology is structured by tree-based architecture then nodes will be hierarchical and fast connection nodes will be grouped together. Figure 6.5 (a) shows the graph structure and network latency values between nodes. Figure 6.5 (b) displays the final result after removing high network latency links from the graph structure. Node A creates its own tree structure as shown in Figure 6.5 (c) from the graph structure.



(a) (b) (c)

Figure 6.5: Finding alternative paths for high network latency nodes and conversion from graph structure to tree structure for node A

As can be seen in Figure 6.5, the longest path from node A to other nodes is the link between A and D which takes 20 ms. Therefore, we can create a A – C – E path with the total time from node A to E of 10 ms, which is half of 20 ms (From A to D). Furthermore, the new path reduces network latency by 20 ms since the network latency from node A to E is 30 ms if node A sends packets directly to node E.

The most important aspect is that it can increase overall game speed by reducing the network latency of the slowest link in the network topology. As can be seen from Figure 6.5, since the B-D connection is the slowest link and the B-A-D link can reduce network latency by 50ms. For graph to tree conversion, one of the most well-known shortest path finding algorithms, Dijkstra (1959) algorithm, is used.

6.3.2. Relay Method

Nodes in this tree-based P2P system may need to relay packets to its child nodes because sender nodes may not have direct connection to destination nodes. To do relay process, the sender node becomes a root node and transmits packets to its own child nodes which are only one level below them. The sender node includes the

identification numbers (id) of destination nodes in its packet header. When its child nodes receive the packet they remove their id from the packet header and relay this packet to their child nodes.

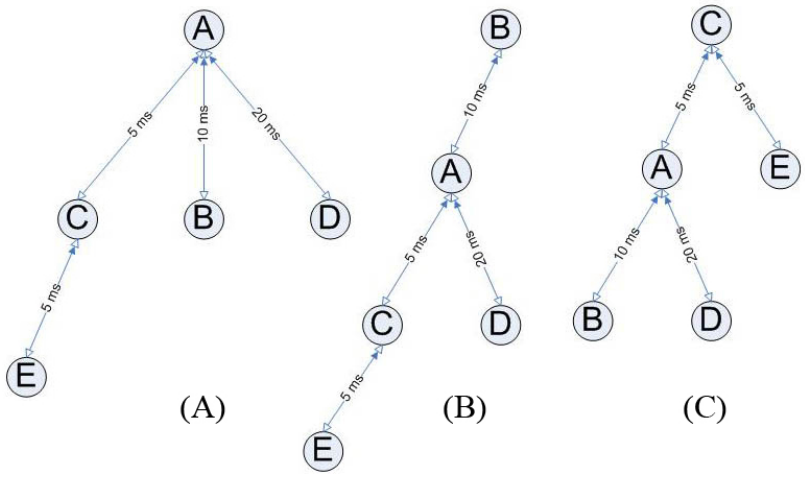


Figure 6.6: Tree Structures for each node (Node A, B, and C only)

Figure 6.6 shows the tree structure for each node. When node A sends a packet to other nodes, it includes the ids of destination nodes. To send a packet From A to E, the destination node list will be [C, E]. Because Node B and D do not have a child node, destination node lists will be [B] and [D] respectively. When node C receives the packet from node A, it removes itself from the destination node list and relays the packet to node E only because node A is not in the destination node list. The pseudo codes below show how the relay algorithm works.

```
/*
Pre-condition:
   nodeNum: A
   origin node list: originNodeList
   desitination node list: destNodeList
Post-condition:
   packets are relayed to immediate child nodes
*/

// Frame Packet Relay Algorithm
relayFramePacket ()
   tempDestNodeList = [ ];
   add A into originNodeList;
   delete A from destinationNodeList;
   for each node in destinationNodeList C
      recursively visit each node in  destinationNodeList
         if each node is in destinationNodeList
            add the node into tempDestNodeList
         end if
      send a packet with tempDestNodeList and orginNodeList to C;
  end for
end relayFramePacket;
```

6.3.3. Retransmission

Retransmission is not avoidable when packet drop or errors occur, especially when frequent state regeneration (FSR) algorithm is not used. The decision for retransmissions is based on ACK timeout and ACK numbers from opponent nodes. The retransmission procedure is exactly same as sending packets at the first time except that the retransmission will occur at the parent node of the node that requests the missing packets.

6.3.4. Acknowledgement

This tree based P2P system can prevent ACK implosion problem because, each node manages its own child or children. When a packet is sent, the destination node sends ACK to its source node immediately. The sender retransmit the packet if

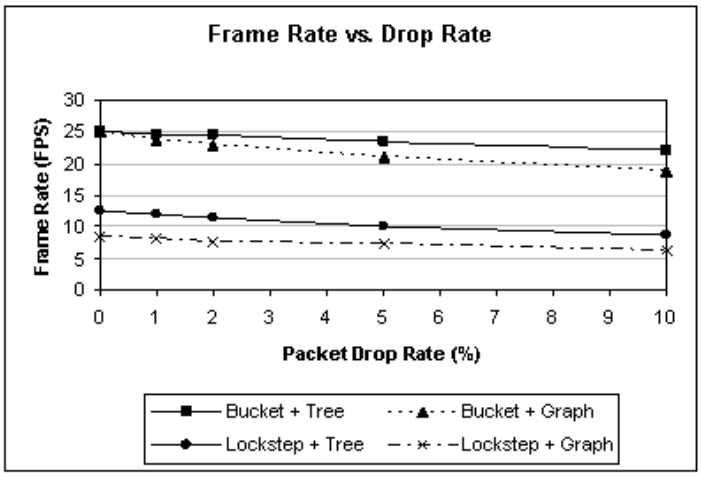
(ACK_timeout_of_destination node < Current_Waiting_Time_For_ACK) is true. The ACK timeout is based on ping time between the two nodes.

6.3.5. Experimental Results

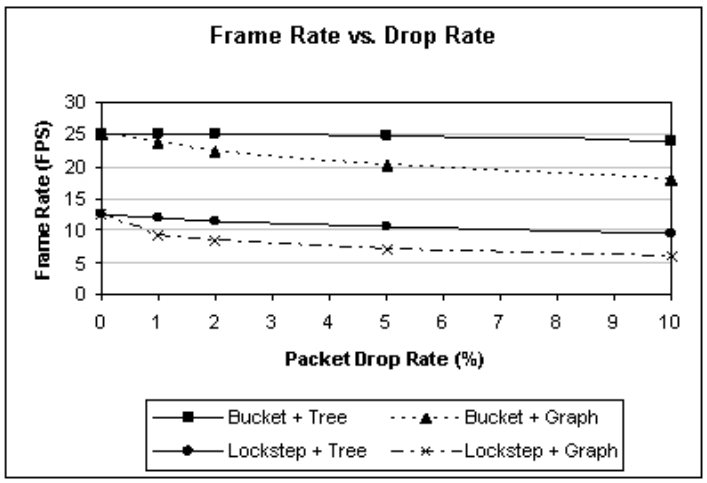
Frame rate and average latency were recorded on four different systems which are combined using two consistency maintenance algorithms (LS and LBS) and two network types (graph-based, and tree-based). These processes were repeated to collect experimental results from three data sets stated in Section 4.1.5.

Figure 6.7 shows the changes in frame rates with respect to drop rate in the four systems on the game sessions of 4, 8, and 16 players. When the packet drop rate is 0, the frame rates of lockstep with graph-based system are 8.35, 12.51 and 8.35 for the number of players 4, 8, and 16 respectively. It is mainly related to frame interval and maximum latency. Current frame interval is 40ms and the maximum latency of the four-player-game session is 82ms. Because lockstep algorithm holds game process until the arrival of all packets from other players and frame packet transmission occurs at every 40 ms, every player must wait for the maximum latency, which is 82ms, and sends packets at every 120 ms. These characteristics of lockstep algorithm explain the frame rates of lockstep with graph-based system under the condition of no packet drop for each case.

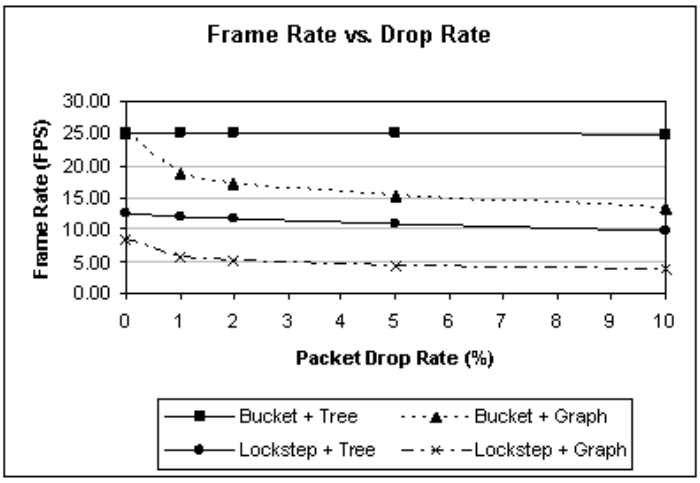
On the other hand, LBS algorithm lets players send their frame packets without waiting for other players' packets until the latency of other packets is less than playout delay which is 120 ms in this occasion. The maximum latency values of three data sets are all less than or equal to 100 ms, thus the optimal frame rate value is determined as 25 fps.



(a) 4 players



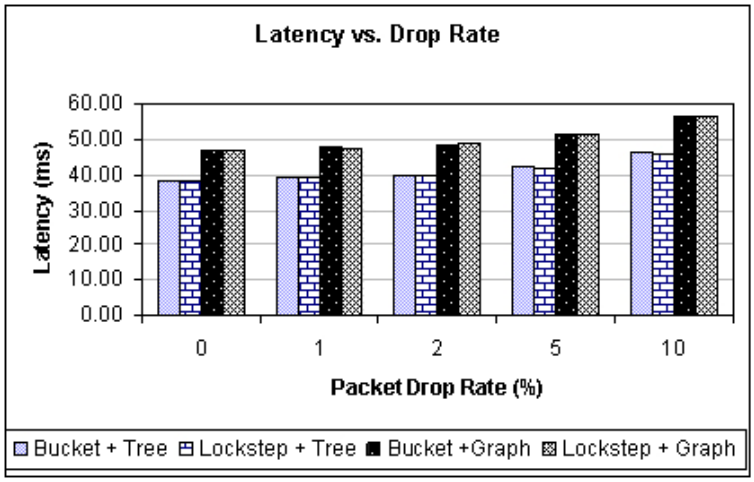
(b) 8 players



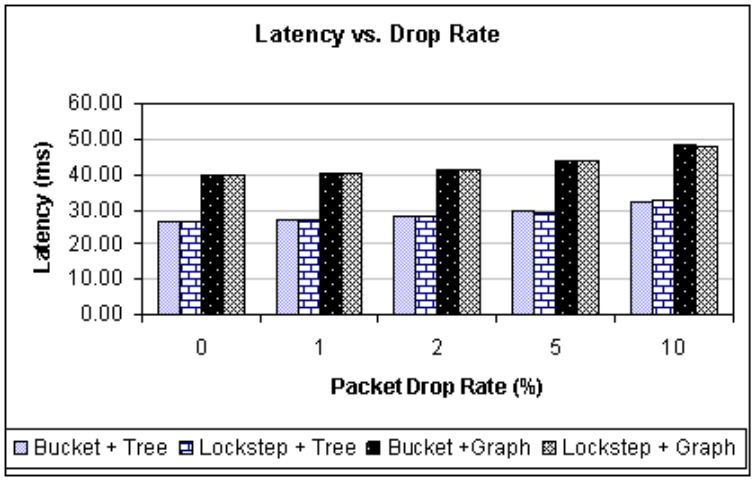
(c) 16 players

Figure 6.7: Frame Rate vs. Drop Rate (4, 8, and 16 players)

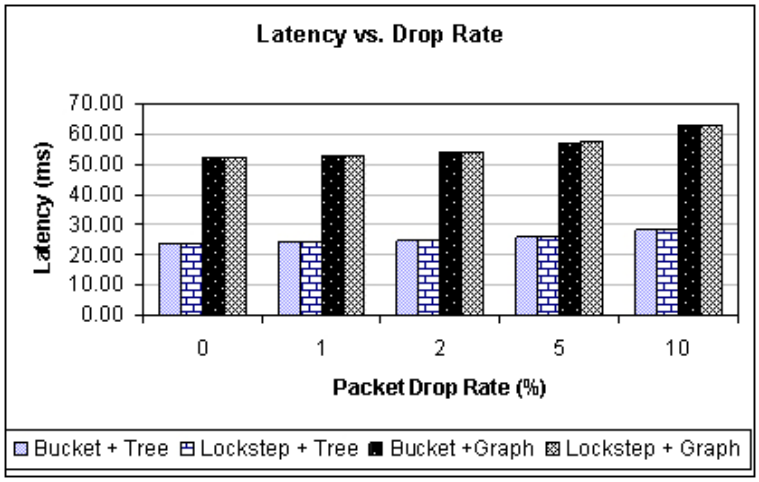
The graphs for the average packet propagation time with packet drop rate on the four combinations are shown in Figure 6.8. As can be seen from the figures, the latency of the two graph-based approaches is similar and so is the latency of the two tree-based approaches. The graph to tree conversion improves the network delay from 46.82 to 38.33ms, from 39.74 to 26.93ms, and from 52.09 to 23.51ms in 4, 8 and 16 player-game sessions respectively.



(a) 4 players



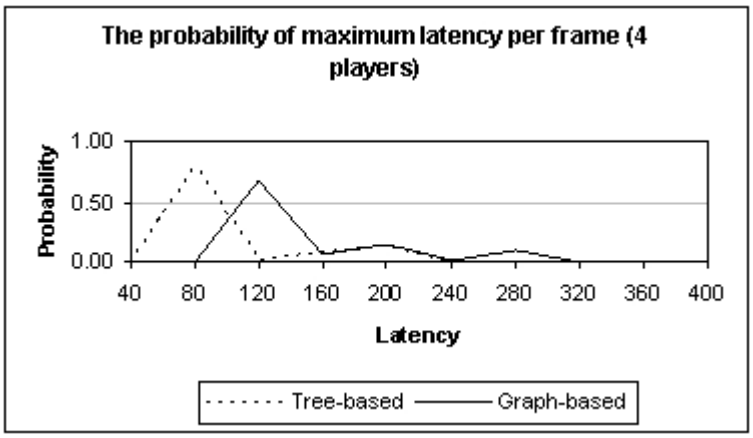
(b) 8 players



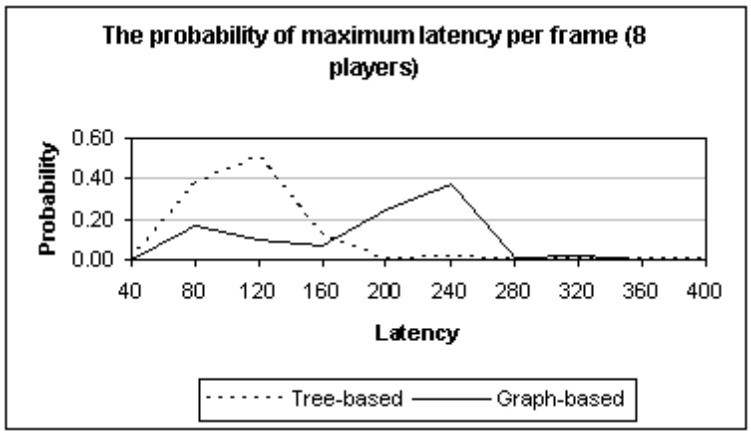
(c) 16 players

Figure 6.8: Latency vs. Drop Rate (4, 8, and 16 players)

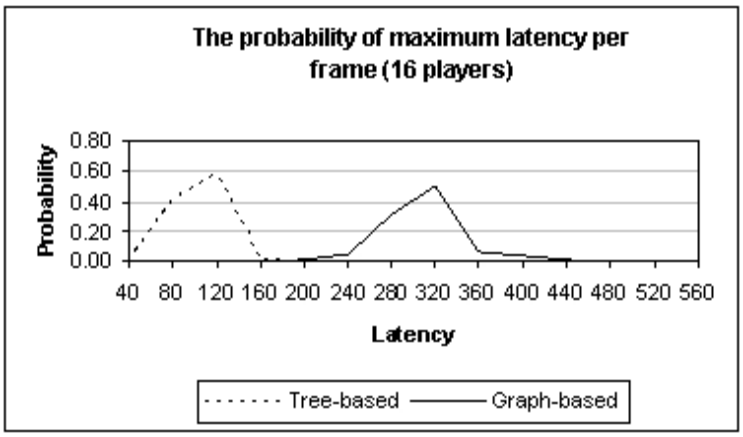
Figure 6.9 shows the probability of maximum frame latency in each frame on two cases, graph-based and tree-based P2P network systems, with the LBS approach. The experimental parameters are 5% packet drop rate, 120 ms playout delay and 40 ms frame interval. The average delayed frame rate changed from 9.24% to 3.02%, from 14.58% to 1.58%, and from 37.81% to 0.12% with graph-based P2P network and tree-based P2P network in 4, 8 and 16 player-game sessions respectively.



(a) 4 players



(b) 8 players



(c) 16 players

Figure 6.9: The probability of maximum latency in each frame on the two approaches with parameters 5% drop rate, 120 ms playout delay and 40 ms frame interval

Figure 6.10 shows the percentage of improvement in frame rate in LBS with tree-based approach. The values are defined as the ratio of improvement of frame rate with respect to the frame rate of the tree-based P2P network system with four-player-game session. According to Figure 6.7, when the number of players increases, frame rate decreases drastically in LBS with graph-based approach. However, seen in Figure 6.10, the LBS with tree-based approach does not degrade the frame rate when the number of players increases, rather it performs better.

The percentage of improvement in average latency is displayed in Figure 6.11. The figure shows that the average latency is improved as the number of players increases. Also, packet drop rate has no effect on the percentage of improvement in average latency.

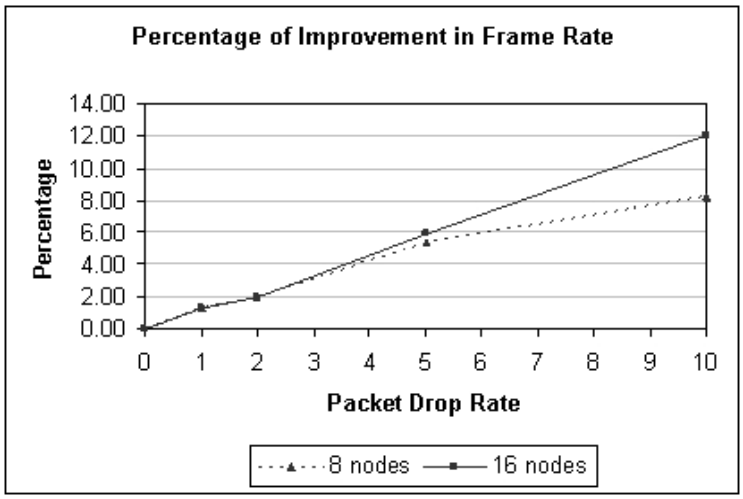


Figure 6.10: Percentage of Improvement in Frame Rate

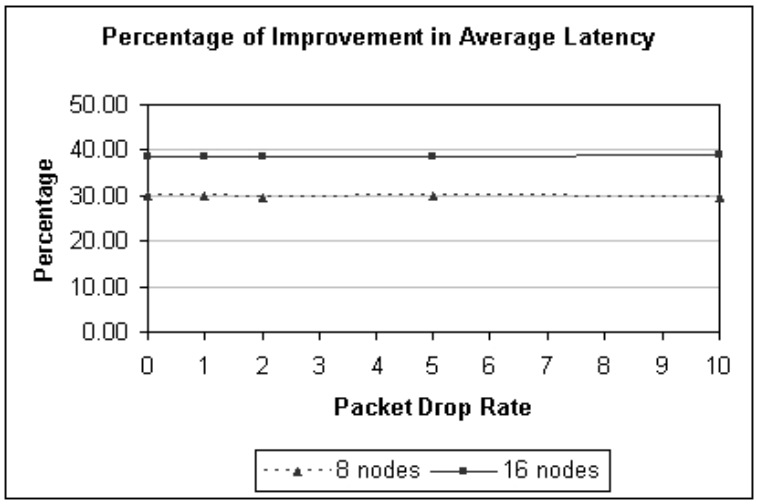


Figure 6.11: Percentage of Improvement in Average Latency

6.3.6. Analysis

Shown from Figure 6.7, the frame rates of game sessions drop significantly when the number of players increases in graph-based network systems. On the other hand, the change rate of the frame rate of game sessions becomes lower as the number of players increases in tree-based network systems and it is evident from Figure 6.10. We assume that this is because the more the number of players the higher the conversion optimality is.

Figure 6.8 also shows similar effects on the number of players on network latency. According to the graphs, the network latency is reduced more as the number of players increases in graph to tree conversion. It is also evident from Figure 6.9. The shape of tree-based graph is similar to graph-based one in each session. We assume that this is because the reduced network latency is the sole parameter that is changed in this experiment. The more the number of players, the greater the distance of parallel displacement from the chart of graph-based system is. Also, this distance is related to the reduction rate of network latency.

6.4. Packet Aggregation Method

In this section, some methods to aggregate information packets and reduce the bandwidth requirements of tree-based P2P network games are examined, with a view to improving system performance.

6.4.1. Bandwidth Requirements of Tree-based Network Games

6.4.1.1. Transmission of information under graph vs. tree structures

In a graph (mesh) structure, each node has direct links to all other nodes and transmits its own status *n-1* times, where *n* is the number of nodes in the game session. Assuming that conservative consistency maintenance algorithms are adopted to ensure consistency among the players, so that information about each frame has to be sent during the game session, the number of total packets to be sent for one frame can be obtained by the formula *n(n-1)*.

The total number of packets to be transmitted for each frame in a tree structure is identical to the one given above. However, because indirect links as well as direct links are used in a tree structure, some packets need to be relayed, which means some players transmit not only their own packets but also other players' packets. Thus, the transmitted number of packets per node is uniform for each node in the graph structure, but may vary quite substantially from node to node in the tree structure.

6.4.1.2. Packet relay

Packet relay does not happen in graph structure, which means each node sends its own packets only and the number of packets per frame is a linear function of the total number of nodes. Graph structure is a special case of tree structure which has no indirect links between nodes. It means that the number of non-immediate child nodes is zero and, therefore, there is no packet replay between nodes.

The total number of packets per frame can be divided into two categories, such as the number of transmitted and received packets per frame. The number of received

packets per frame (NRPF) in mesh and tree structure is identical because the packets are from each node. The number of transmitted packets per frame (NTPF) is, however, different for different nodes in tree structure due to packet relay. Assuming there is no direct link between immediate child nodes in the tree structure of node A, the NTPF for node A can be calculated using Eq. (6.1). This equation can be expressed with the number of total nodes n and the number of immediate child nodes, yielding Eq. (6.2).

$$NTPF(A) = \alpha^2 + \beta(\alpha - 1) \quad (6.1)$$

$$NTPF(A) = n(\alpha - 1) + 1 \quad (6.2)$$

n: the total number of nodes

α: the number of immediate child nodes

β: the number of non-immediate child nodes.

As can be seen from the above equations, NTPF(A) is a quadratic function of α, the number of immediate child nodes (ICN) in the worst case where ICNs do not have direct links between them. If ICN is 1 then NTPF(A) is 1 according to the equations and it means high reduction rate of bandwidth requirement for the node. However, the node which has direct links to all other nodes should transmit packets $(n-1)^2$ times. This fact implies that bandwidth requirement of the node and overhead balancing are important factors to consider when establishing tree structure P2P systems.

6.4.2. Packet Aggregation

When packet-relay happens, the relay-node immediately sends the packet to reduce network latency. Also, the node needs to send its own packets to the destination.

Assuming we use conservative consistency algorithms, there will be no additional delay if the node waits for relay-packets and aggregate all packets including its own packets for sending to each destination. This approach can reduce bandwidth requirement of the node by decreasing NTPF of the node. When a packet is sent, it is transmitted with additional information which is called packet header. This packet aggregation method can reduce the number of packet headers. Therefore it reduces the bandwidth requirement of the node.

The best case of packet aggregation is that a shortest path exists which goes through all nodes. For example, if there are 3 nodes then A-B-C route is shorter than A-C. In this case, node B will be a super node that connects all other nodes. In the best case, the number of total packets per frame in tree structure with aggregation method can be obtained using the expression $2(n-1)$ where n is the number of nodes. In the worst case, it will be $n(n-1)$ and this is the case of mesh structure and there is no packet aggregation at all. We can write an expression to indicate the average value by adding the above two expressions and dividing them by 2. Eq. (6.3) shows the detail.

$$(n(n-1) + 2(n-1)) / 2 = ((n-1)(n+2))/2 \qquad (6.3)$$

The number of packets per frame in tree structure is always smaller than in mesh structure when n is greater than 3, as can be seen in Eq. 6.3. Generally, P2P requires more than three nodes. Therefore, we can guarantee that the packet aggregation method reduces bandwidth requirement in tree structure relative to relay-only method in any case. The number of packets per frame is a linear function of the number of nodes in the best case and it is a quadratic function in the worst case.

6.4.3. Experimental Results

Two consistency maintenance algorithms, Lockstep (LS) and Locked Bucket-Synchronization algorithm (LBS), are implemented for this experiment. The experiment duration is 60 seconds and frame interval is 100 ms because the maximum network latency is set as 100 ms between players. Playout delay for LBS is 200 ms and two network structures are used, namely graph and tree.

Table 6.2, Figure 6.12 and Figure 6.13 show experimental results based on LS and LBS algorithms under graph structure. The first row in the table shows the optimal results of the game session which has no packet drop at all. Therefore, FPS (Frame per Second) value is 10 and this value is the same as the optimal value. In optimal situation, the formula for the total number of packets per frame is expressed as $n(n-1)\times 2$ where n is the number of players. A factor of 2 is included because we also count acknowledgement packets even though their size is smaller than the size of frame packets. Therefore, the value for the Average Packets per Frame for eight-player game sessions is 112 as shown in the table. Figure 6.14 and Figure 6.15 display the results of experiments using the two algorithms under tree structure.

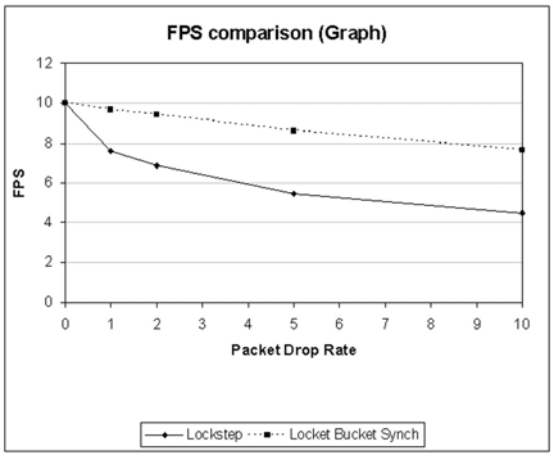


Figure 6.12: the changes of FPS according to packet drop rate on graph structure

Table 6.2: Experimental results of Lockstep algorithm on graph structure

Drop Rate (%)	Total Packets	FPS	Avg. Packets per Frame
0	67066	10.00	111.78
1	51612	7.58	113.43
2	47517	6.88	115.05
5	39398	5.47	120.12
10	35410	4.52	130.66

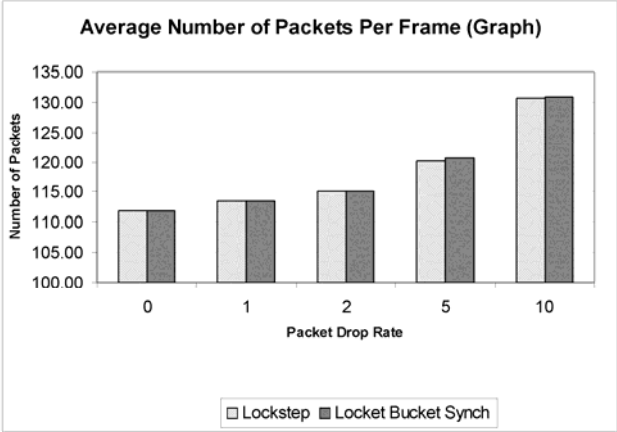


Figure 6.13: the changes of average number of packets per frame according to packet drop rate on graph structure

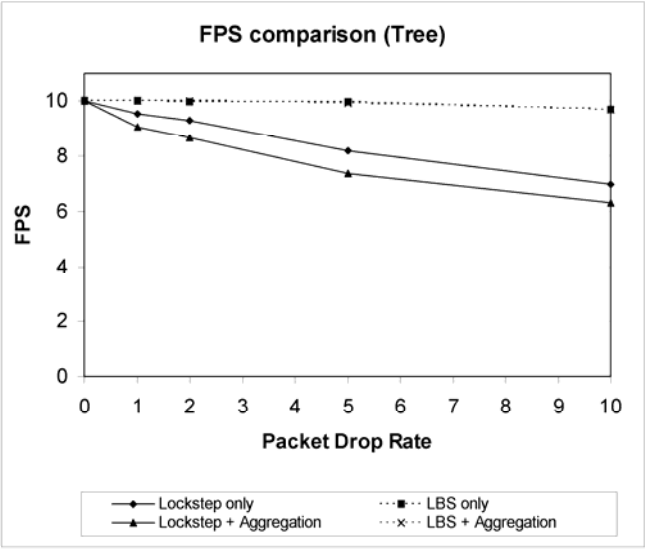


Figure 6.14: The changes of FPS according to packet drop rate on tree structure

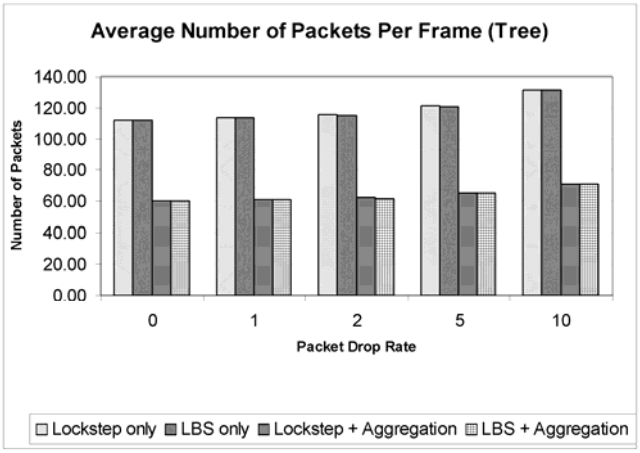


Figure 6.15: The changes of average number of packets per frame according to packet drop rate on tree structure

6.4.4. Analysis

As shown in Figure 6.12, LBS algorithm performs better than LS on packet drop rate, in terms of game execution speed measured by FPS. LS algorithm achieved around 45% of optimal value and LBS algorithm achieved about 76% on 10% packet drop rate. The average numbers of packets per frame under graph structure are almost identical for both algorithms as displayed in Figure 6.13. This implies that LBS algorithm performs better than LS algorithm in terms of game execution speed without increasing bandwidth requirement for players.

To reduce network latency and bandwidth requirement, tree-based P2P system is introduced and the experimental results are shown in Figure 6.14 and Figure 6.15. The FPS values are almost optimal (about 97%) when LBS algorithm is employed on tree structure even with 10% packet drop rate as shown in Figure 6.14. The numbers of packets per frame for each algorithm on tree structure are almost identical. However, when aggregation method is utilized, the number of packets per frame is dramatically reduced from 112 to 60 on 0% of packet drop rate. The number of packets per frame

for eight-player game sessions is 63 when packet aggregation is available in average case according to Eq. (3). The number becomes 112 in the worst case as clearly shown in Figure 6.15.

6.5. Chapter Summary

Network latency and bandwidth requirement are important factors affecting the playability of network games. A major objective of this chapter is to alleviate latency by reducing delays caused by dropped packets -- this is to be achieved through the use of special transmission schemes. In Section 6.2, the performance of the smart transmission scheme (STS) was compared with those of the blind transmission scheme (BTS) and no special transmission scheme (NTS). The comparisons were made in association with two conservative consistency maintenance algorithms, Lockstep (LS) and Locked Bucket Synchronization (LBS), both of which had been examined in Chapter 5. The experiments were conducted by means of the COMP2P network simulator with the network settings of 4, 8, and 16 player nodes.

The experimental results indicate that if the packet drop rate is zero, the LS algorithm with NTS produces excellent performances in terms of game execution speed -- indeed, this speed reaches 10 fps (frames per second), which is the maximum possible execution speed in the current circumstances, regardless of whether there are 4, 8 or 16 players. As the packet drop rate increases, however, the execution speed of the LS algorithm decreases sharply. For example, if the drop rate is 10%, this speed is reduced to 6.05 fps, 4.52 fps, and 3.00 fps for the 4-player, 8-player, and 16-player cases, respectively.

The introduction of BTS raises the corresponding execution speed to 9.02 fps, 7.47 fps, and 5.08 fps, respectively. As can be expected, however, there is a trade-off between game execution speed and bandwidth requirements. For the representative case of 8 players, for example, the LS-BTS combination requires 1,621 pps (packets to be sent per second), i.e., more than double the 591 pps required by the LS-NTS combination. Similar results apply for the other numbers of players. The differences in bandwidth requirements can be explained largely in terms of the need to transmit information twice.

The increase in bandwidth requirement associated with BTS may make it unacceptable in the context of some genre of multiplayer network games. Therefore, the combination of the LS algorithm with STS is proposed, and the experimental results confirm that, compared with BTS, STS reduces bandwidth requirements substantially while sacrificing relatively little in terms of game speed, especially when the drop rate is non-negligible. At 10% drop rate, for example, the LS-STS combination for 8-node case achieves a game execution speed of 7.08 fps (only 5% lower than LS-BTS) while requiring 1,253 pps (23% lower than LS-BTS).

Continuing with the 10% drop rate, the use of the more sophisticated LBS algorithm results in a sizable improvement over the LS algorithm in terms of execution speed: in the 8-node case, the LBS-NTS combination achieves a game speed of 7.65 fps (69% higher than LS-NTS). It also requires greater bandwidth: 1,001 pps (69% higher than LS-NTS). Similar to the case of the LS algorithm, combining the LBS algorithm with BTS improves game speed considerably (to 9.47 fps) but also more than doubles the bandwidth requirements (to 2,053 pps). Once again, STS sharply reduces the latter requirement (to 1,300 pps) while reducing game speed only slightly (to 8.90 fps).

The second major objective of this chapter is to reduce latency by adopting a tree-based network structure. This theme is developed in Section 6.3. The major advantage of the tree-based network structure compared to a graph-based one is the utilisation of faster and broader connections between some nodes. The faster connections can be used to shorten network latency between a given pair of nodes by finding alternative, faster paths rather than the most direct path. Similarly, the broader connections allow alternative paths which can provide larger bandwidth than the most direct path. In the current research, only the utilisation of faster connections has been adopted.

Packet drop rate is one of the most important factors which can cause network latency. Therefore, we investigated the relative effects of packet drop on the two alternative approaches to peer-to-peer network structuring. Two consistency maintenance algorithms, LS and LBS, were implemented to analyse the efficiency of the tree-based approach and compare it with the graph-based approach, under the conditions of three types of P2P game sessions: small, medium and large (as exemplified by 4-node, 8-node and 16-node sessions).

Under identical conditions as before (10% packet drop rate, 8 nodes) the LS-NTS combination achieves an execution speed of 6.98 fps and requires 918 pps when the tree structure is used, compared with 4.52 fps and 591 pps for the graph structure. Not surprisingly, the adoption of the LBS (in place of LS) algorithm improves the performance further: the LBS-NTS combination attains a game speed of 9.72 fps with a requirement of 1,277 pps under the tree structure.

Thus the experiments indicate the superiority of the tree-based over the graph based approach in many circumstances, especially when the packet drop rate is relatively

high. It is interesting to note also that, in the experiments conducted for this study, as the number of players increased, this superiority did not diminish but tended rather to increase. This may be due the fact that as the number of nodes increased, there was greater scope for efficiencies to be gained through the use of faster, non-direct paths.

In short, the experimental results suggest that the tree-based network system with LBS algorithm is suitable for Internet games which suffer from unpredictable network latency, jitter, and packet drops. It is believed that the superiority of the tree-based system is based on not only the reduction of latency from the graph to tree conversion, but also on the latter's relay mechanism.

A third major objective of this chapter is to reduce the bandwidth requirements of tree-based P2P MODIGs through packet aggregation -- this innovation is reported in Section 6.4. In a tree-based structure, the system finds the shortest path between each pair of players in order to reduce network latency. If indirect links are used, information packets need to be relayed, which means some players need to transmit not only their own packets but also other players' packets. Normally, when such packet-relays are required, as soon as the relay-node receives a packet to be relayed, it immediately sends the packet on. Assuming the use of conservative consistency algorithms, however, there will be no additional delay to the system overall if the node waits for other relay-packets and aggregates all packets including its own packets before sending all to the destination nodes.

To examine the effects of the proposed aggregation method on bandwidth requirements, experiments were performed with the use of two conservative consistency maintenance algorithms, LS and LBS, in a tree-based system. The experiments were carried out on the network simulator, COMP2P, previously

described in Section 4.2, with the network setting of 8 player nodes. For brevity, only the NTS scheme was considered.

As noted above, when the packet drop rate is 10%, the LS algorithm with no packet aggregation achieved an execution speed of 6.98 fps, with a bandwidth requirement of 918 pps. If packet aggregation were utilized, the latter requirement would be more than halved, to 444 pps, while the execution speed would change only slightly (to 6.30 fps). Similarly, packet aggregation would almost halve the bandwidth requirement of the LBS algorithm, from 1,277 pps to 682 pps, while reducing its game speed only marginally, from 9.72 fps to 9.68 fps.

The main purpose of these experiments has been to assess the effects of, one by one, each of the proposed innovations. This is why in Section 6.4, only NTS was used in combination with LS/LBS in a tree-based structure. In future research, all of the three innovations proposed in this chapter will be implemented simultaneously, so that STS/BTS would also be used in such combinations.

7. Conclusion

7.1. Summary of Main Findings

There is no doubt that Multiplayer Online Digital Games (MODIGs) stand to gain greater popularity in the future, as the carrying capacity and processing power of computer networks grow. This trend will probably be most apparent in entertainment areas, as interactions (playing games) with actual human players tend to be, at least for some time to come, more complex and therefore more interesting than with virtual, artificial intelligence (AI) players. In addition, MODIGs are likely to be useful in training, education, and academic and military research.

Nevertheless, the actualisation of MODIGs remains difficult because of a range of technical issues. In particular, problems related to networking -- such as limitations in data transfer rate, latency, and jitter -- are among the most difficult to resolve. Network latency cannot be avoided completely and introduces various problems such as inconsistency of player status, recognisable responsiveness, and irregular network lag. In this study, we examined current approaches to maintain consistency and suggested several innovations to alleviate network latency, enhance execution speed and reduce bandwidth requirements.

In general, the network architectures of MODIGs can be classified under three headings: Client-Server (C/S), Peer-to-Peer (P2P), and hybrid. Many MODIG designers prefer the C/S network architecture to the P2P system, because of a number of advantages enjoyed by the former, such as simplicity of consistency maintenance, improved security, efficient authentification, and ease of billing system management. However, the C/S architecture is prone to severe network latency, with servers often

becoming critical bottlenecks. To overcome this problem, server clustering methods are often used, but these solutions may not be very cost-effective.

This is why a number of recently introduced games use the P2P network architecture, but in this structure the total number of players in any one game session is often limited, in line with the network's bandwidth constraints. In addition, the consistency maintenance issue becomes critical within this architecture.

There are two main approaches for maintaining consistency in MODIGs. Under the conservative approach, a send-and-wait philosophy is adopted, thus requiring acknowledgement frames and resulting in packet transfer delay, so that network latency is more likely. Under the optimistic approach, the processes do not wait for other players' packets but continue to advance to their own frames, and so there may be little or no network latency. However, when there is inconsistency between players, the processes must roll back to correct mis-ordered operations due to packet transfer delay. These can cause irritation and confusion to players, and thus the quality of game deteriorates. Overall, the optimistic approach may not be sufficiently robust for actual network games with a reasonably large number of players.

To alleviate network latency and reduce the bandwidth requirements of conservative consistency maintenance algorithms under a P2P architecture, a number of innovations are proposed and designed. First, to reduce network latency, a conservative consistency maintenance algorithm named Locked Bucket Synchronisation (LBS) is proposed: it masks latency and maintains perfect consistency among players. Second, to alleviate network latency caused by packet drop and delay, a smart transmission scheme (STS) is proposed. Third, a distributed network architecture is adopted and a tree-based P2P system is proposed, to remove

additional packet transfer delays between server and client. Fourth, a packet aggregation method is introduced to reduce bandwidth requirements.

These proposed innovations are implemented and their performances are thoroughly examined and compared. In particular, a network simulator, COMP2P, and a real network game, Duel-X, are implemented to serve as testing beds for the proposed innovations. The system architectures of COMP2P and Duel-X as well as the results of the comparative experiments are documented.

Locked Bucket Synchronisation

The efficiency of the proposed consistency algorithm, LBS, is compared with that of other approaches such as Lockstep (LS) and Frequent State Regeneration (FSR). The experimental results from COMP2P show that the proposed LBS algorithm outperforms the LS algorithm under all tested circumstances, in terms of game execution speed. For example, the LBS algorithm with the tree-based P2P system achieves near-optimal frame rate under the condition of a 10% packet drop rate, while the LS algorithm with the tree-base P2P system performs at approximately 40% of optimal frame rate.

Smart Transmission Scheme

The efficiency of the Smart Transmission Scheme (STS), as applied to the LBS and competing algorithms, is compared with the Blind Transmission Scheme (BTS) and the No Special Transmission Scheme (NTS). Under NTS, packets are routinely transmitted only once between each relevant pair of nodes, but when a packet is dropped, it must be resent. This implies that a dropped game packet can cause a minimum delay of $3n$, where n denotes the delay in packet propagation from node A

to node B. To minimise the effect of packet drop and delay, game packets can be sent twice in every instance; this is known as the "Blind" BTS. By contrast, under the "Smart" STS, a packet is sent twice only if it is critical, with the criticality of packets being determined by the latency between nodes and playout delay.

The results of experiments with COMP2P and LBS indicate that BTS increases the speed of game execution relative to that of NTS, but not surprisingly BTS also requires greater bandwidth -- indeed, the number of packets transmitted per frame is increased by up to 100%. By contrast, STS almost matches BTS in terms of game execution speed, but requires only about 10% more packets to be transmitted per frame.

Tree-based P2P System

The tree-based P2P system searches for the shortest paths, in terms of packet transfer costs, between nodes which participate in game sessions, in order to reduce latency and/or bandwidth requirements of particular nodes. One possible criterion to be used in this search is to measure packet transfer cost in terms of network latency. In this case, the tree-based P2P system searches for the path which yields the shortest network latency between origin and destination nodes. Routers in the Internet themselves routinely utilise various path finding algorithms. However, it is impractical to establish all possible connections between all routers. Therefore, the efficiency of routing algorithms relies mainly on the connections between routers in *neighbouring* networks. If a direct path between two nodes consists of poorly managed or latency-prone routers, then there may be indirect paths that are more efficient because they bypass these routers. An alternative method to decide the topology of P2P systems is to use bandwidth constraints (relative lack of spare

bandwidth capacity) as a measure of transfer costs. In this case, the tree-based P2P system searches for the path between origin and destination nodes which minimises the involvement of nodes with known severe bandwidth constraints.

A team-play option is one of the possible match types in MODIGs, where some participants in the game session may play with other participants in the same, or a neighbouring, network. In this case, the situation essentially reduces to playing online games in a local network setting. In general, LAN-based packet transmission is more reliable than WAN-based transmission, in terms of both network latency and bandwidth.

Results from the various experiments using COMP2P and LBS show that the tree-based P2P system performs better than the corresponding graph-based system under all of the circumstances considered. However, the tree-based architecture aggravates the bandwidth requirements of some parent nodes.

Packet Aggregation

Finally, the proposed packet aggregation method, when applied in conjunction with the LBS algorithm, reduces by about 50% the bandwidth requirement of players, without lowering game execution speed, in comparison with the case of no packet aggregation (that is, packets being sent without waiting to be aggregated).

Real-time Game Situations

In the discussion above, the efficiency of the various innovations is observed through simulations with the COMP2P simulator. To further verify and validate these results, various experiments are performed using the actual online game, Duel-X. The experimental results indicate that, by alleviating latency without degrading

consistency between players, the LBS algorithm can improve the playability of MODIGs considerably. Indeed, because of imperfect human perceptive capabilities, network latency is effectively masked (not perceived by players) under LBS.

Overall, the experimental results show that actualisation of reliable and robust MODIGs is achievable with a combination of P2P-based architectures and conservative consistency maintenance algorithms.

7.2. Limitations and Further Research Directions

Tree-based Structures

Our experimental results suggest that tree-based peer-to-peer networks may be more appropriate for the Internet games than graph-based ones. It is recognised, however, that the tree-based network structure may suffer from several limitations, such as a lack of flexibility with respect to node-join/drop during runtime, an insufficiency of alternative paths, and the complexity of the system.

The lack of flexibility regarding node-join and drop means that the structure should be changed whenever new nodes join the system and/or existing nodes are dropped from the game session. In addition, this change should be notified to all other nodes so that they can refresh their network structure information.

The insufficiency of alternative paths is closely related to the efficiency of current Internet routers. Our preliminary experiments suggest that, in real world situations, it may be difficult to find alternative paths between two nodes which are shorter (less costly) than the most direct path by a margin of 10% or more.

The complexity of the system is increased by a particular characteristic of packet dissemination in tree-based structures, namely the relay of some information packets via indirect paths, as opposed to the once-only transmission over a direct path. For efficient packet relay, each node must know the global network topology which should be synchronised among all participants.

These limitations are beyond the scope of the current study but should be addressed in future research.

Packet Aggregation

In examining the effects of the aggregation method on the two conservative consistency maintenance algorithms, LS and LBS, experiments were performed by using COMP2P with the network setting of 8 player nodes. The LBS algorithm with aggregation method performs better than any other combination in terms of game frame rate and bandwidth requirement. However, LBS prolongs playout-delay which also affects game playability and this may be critical to some genres of network games. Therefore, the effects of the playout-delay on distributed network games should be examined in further research. In addition, it will be interesting to vary the number of players and observe the impact (if any) on the efficiency of the packet aggregation method.

Real-time Game Situations

It is recognised that many of the results reported in this study have not been obtained from truly real-world situations, but rather from simulations. For example, the experimental data for latency between nodes are often artificial (randomly generated for use with the COMP2P simulator). Similarly, the reductions in bandwidth

requirements that become available through the use of packet aggregation in a tree-based network structure are obtained via simulations. It will be worthwhile to replicate the various experiments under actual rather than artificial conditions in future research. Moreover, it will be desirable to vary more than one factor (e.g., transmission scheme, tree/graph structure, aggregation) at a time. These are some of the directions that the author intends to pursue in future research.

Other Factors

While latency and bandwidth have been the main focal points of this study, it is recognised that other factors also play important roles in determining the playability and user acceptance of a MODIG. Examples include processing power, security, cheating prevention, accessibility, cost, aesthetics, and (above all) the attraction of the game's central premise or idea.

Appendix A

COMP2P Network Simulator

graphToTreeSimulator.h

```
#include <stdarg.h>

const int MAX_CHAR = 256;
const int MAX_NODE_NUM = 100;
const int WIDTH = 8;

class GraphToTreeSimulator
{
public:
    GraphToTreeSimulator();
    ~GraphToTreeSimulator();

    void    setExperimentNum(int iExperimentNum) { m_iExperimentNum =
iExperimentNum; }
    int getExperimentNum() { return m_iExperimentNum; }

    void    setGraphTypeArray(int iCount, ...);
    int*    getGraphTypeArray() { return m_pGraphTypeArray; }
    int     getGraphTypeNum() { return m_iGraphTypeNum; }

    void    setSimulationTime(int iSimulationTime) { m_iSimulationTime =
iSimulationTime; }
    int     getSimulationTime() { return m_iSimulationTime; }

    void    setAlgorithmArray(int iCount, ...);
    int*    getAlgorithmArray() { return m_pAlgorithmArray; }
    int     getAlgorithmNum() { return m_iAlgorithmNum; }

    void    setFrameInterval(int iFrameInterval) { m_iFrameInterval =
iFrameInterval; }
    int     getFrameInterval() { return m_iFrameInterval; }

    void    setPlayoutDelayArray(int iCount, ...);
    int*    getPlayoutDelayArray() { return m_pPlayoutDelayArray; }
    int     getPlayoutDelayNum() { return m_iPlayoutDelayNum; }

    void    setPacketDropRateArray(int iCount, ...);
    int*    getPacketDropRateArray() { return m_pPacketDropRateArray; }
    int     getPacketDropRateNum() { return m_iPacketDropRateNum; }

    void   setDataFileName(string strFileName) { m_strDataFileName = strFileName; }

    void    run();
```

```
    void    setGraph(int iGraphType);

    void    setupAggregationLists();

    bool    readTotalNodeNum(ifstream& infile);

    bool    startSimulation(int iPacketDropRate);

    bool    createOutputFiles(char *strParameters, char *strDataDirectory);

    void    cleanUpAll();

    void    printAllPlayersPath();

    void    setAggregation(bool bAggregation) {m_bAggregation = bAggregation;};
    void    setAggregationScheme(int iAggregationScheme) { m_iAggregationScheme =
iAggregationScheme;};
    void    setRetransmissionScheme(int iRetransmission) { m_iRetransmission =
iRetransmission;};

private:
    // called from the setupAggregationLists method
    void    passAllPaths();

    // the number of experiment
    int m_iExperimentNum;

    // Graph type (0:connected graph, 1:tree-based)
    int m_iGraphType;
    int m_iGraphTypeNum;
    int *m_pGraphTypeArray;

    // simulation time in minutes
    int m_iSimulationTime;

    // packet generation algorithms
    int m_iAlgorithm;

    // 0: time-based
    // 1: lock-step
    // 2: bucket-synchronisation
    int m_iAlgorithmNum;
    int *m_pAlgorithmArray;

    // frame interval in milli-seconds
    // for example, if iFrameInterval is 30 then
    // each node's frame will move forward at
    // every 30 milli-seconds.
    int m_iFrameInterval;

    // playout delay in milli-seconds
    // for a bucket-synchronisation algorithm.
```

```
    int m_iPlayoutDelay;
    int m_iPlayoutDelayNum;
    int *m_pPlayoutDelayArray;

    // packet drop rate in percentage (whole number)
    int m_iPacketDropRate;
    int m_iPacketDropRateNum;
    int *m_pPacketDropRateArray;

    bool    m_bAggregation;
    int m_iAggregationScheme;
    int m_iRetransmission;

    string m_strDataFileName;

    int m_iTotalNodeNum;

    playerNode<int>* m_players;
    weightedGraphType<int, MAX_NODE_NUM> m_graph;
    Statistics m_statistics;

    // output file stream for storing packet data
    ofstream *m_outfile;

    // private methods
    void doItNow(int h, int i, int j, int k, int l, char *strDirectory, bool
bAggregation);

};

GraphToTreeSimulator::GraphToTreeSimulator()
{
    m_iExperimentNum = 0;

    m_iGraphType = 0;
    m_iGraphTypeNum = 0;
    m_pGraphTypeArray = NULL;

    m_iAlgorithm = 0;
    m_iAlgorithmNum = 0;
    m_pAlgorithmArray = NULL;

    m_iPlayoutDelay = 0;
    m_iPlayoutDelayNum = 0;
    m_pPlayoutDelayArray = NULL;

    m_iPacketDropRate = 0;
    m_iPacketDropRateNum = 0;
    m_pPacketDropRateArray = NULL;

    m_iFrameInterval = 0;
    m_iSimulationTime = 0;
```

```
    m_strDataFileName = "";

    m_players = NULL;
    m_outfile = NULL;

    m_iTotalNodeNum = 0;

    m_bAggregation = false;
    m_iRetransmission = 0;
}

GraphToTreeSimulator::~GraphToTreeSimulator()
{
    delete [] m_pGraphTypeArray;
    delete [] m_pAlgorithmArray;
    delete [] m_pPacketDropRateArray;

    delete [] m_players;
    delete [] m_outfile;
}

void GraphToTreeSimulator::setGraphTypeArray(int iCount, ...)
{
    m_pGraphTypeArray = new int[iCount];
    m_iGraphTypeNum = iCount;

    // will point to each unnamed argument in turn
    va_list ap;

    // point to first element
    va_start(ap, iCount);

    for (int i=0; i<iCount; i++)
    {
        m_pGraphTypeArray[i] = va_arg(ap,int);
    }

    // cleanup
    va_end(ap);
}

void GraphToTreeSimulator::setAlgorithmArray(int iCount, ...)
{
    m_pAlgorithmArray = new int[iCount];
    m_iAlgorithmNum = iCount;

    // will point to each unnamed argument in turn
    va_list ap;

    // point to first element
    va_start(ap, iCount);

    for (int i=0; i<iCount; i++)
```

```
    {
        m_pAlgorithmArray[i] = va_arg(ap,int);
    }

    // cleanup
    va_end(ap);
}

void GraphToTreeSimulator::setPlayoutDelayArray(int iCount, ...)
{
    m_pPlayoutDelayArray = new int[iCount];
    m_iPlayoutDelayNum = iCount;

    // will point to each unnamed argument in turn
    va_list ap;

    // point to first element
    va_start(ap, iCount);

    for (int i=0; i<iCount; i++)
    {
        m_pPlayoutDelayArray[i] = va_arg(ap,int);
    }

    // cleanup
    va_end(ap);
}

void GraphToTreeSimulator::setPacketDropRateArray(int iCount, ...)
{
    m_pPacketDropRateArray = new int[iCount];
    m_iPacketDropRateNum = iCount;

    // will point to each unnamed argument in turn
    va_list ap;

    // point to first element
    va_start(ap, iCount);

    for (int i=0; i<iCount; i++)
    {
        m_pPacketDropRateArray[i] = va_arg(ap,int);
    }

    // cleanup
    va_end(ap);
}

void GraphToTreeSimulator::doItNow(int h, int i, int j, int k, int l, char
*strDirectory, bool bAgg)
{
    char strParameters[MAX_CHAR];
    char strDataDirectory[MAX_CHAR];
```

```
    bool bQuit = false;

    // set the parameter string
    sprintf(strParameters, "%d_%d_%d_%d_%d_%d_%d_%d",
            h+1, m_pGraphTypeArray[i], m_pAlgorithmArray[j],
            m_pPlayoutDelayArray[k], m_pPacketDropRateArray[l],
            m_iSimulationTime, m_iFrameInterval, bAgg);

    // set the data directory string
    sprintf( strDataDirectory, "%s\\%s", strDirectory, strParameters );

    createOutputFiles(strParameters, strDataDirectory);


    // start simulation now!
    cout << "Now, we are starting the simulation. Hit any key to stop" << endl <<
endl;

    bQuit = startSimulation(m_pPacketDropRateArray[l]);

    // save the experiment result into a file
    m_statistics.saveExperimentalResultIntoFile(strDataDirectory);

    // save the parameter set into a file
    m_statistics.saveParametersIntoFile(strDataDirectory, m_pGraphTypeArray[i],
                m_iSimulationTime, m_pAlgorithmArray[j], m_iFrameInterval,
                m_pPlayoutDelayArray[k], m_pPacketDropRateArray[l], bAgg);

    // save the latency report into a file
    m_statistics.saveLatencyReportIntoFile(strDataDirectory);

    // close all outfiles
    for(int m=0; m<=m_iTotalNodeNum; m++)
    {
        m_outfile[m].close();
    }

    cleanUpAll();

    if (!bQuit) exit(0);
}

void GraphToTreeSimulator::run()
{

    char strDirectory[MAX_CHAR];
    bool bAggregation = false;


    ifstream infile;            //input file stream for getting graph data
    __time64_t lTime;

    if (!readTotalNodeNum(infile)) exit(0);         // get the number of nodes
```

```
// for testing
cout << "Total Number of Nodes: " << m_iTotalNodeNum << endl << endl;

// Read Data & Set up the Graph
m_graph.createWeightedGraph(infile, m_iTotalNodeNum);

m_players = new playerNode<int>[m_iTotalNodeNum];

// set aggregation scheme for players
m_players[0].s_iAggregationScheme = m_iAggregationScheme;

// set retransmission scheme for players
m_players[0].s_iRetransmission = m_iRetransmission;

// set simulationTime for players
m_players[0].s_iSimulationTime = m_iSimulationTime;

// set the frame interval for each node
m_players[0].s_iFrameInterval = m_iFrameInterval;

// initialize playerNode
for (int i = 0; i < m_iTotalNodeNum; i++)
{
    m_players[i].setPlayerNum(i, m_iTotalNodeNum);
}

m_outfile = new ofstream[m_iTotalNodeNum + 1];

// create a statistics object
m_statistics.setStatistics( m_iTotalNodeNum,
        (int)(m_iSimulationTime * 1000.0 / m_iFrameInterval) );

for(int h=0; h<m_iExperimentNum; h++)
{
    // get the current time
    _time64( &lTime );

    // Seed the random-number generator with current time
    srand( (unsigned int)lTime );

    sprintf( strDirectory, "Data\\%s_%d", m_strDataFileName.data(), lTime );

    // create a directory
    ::CreateDirectory(strDirectory, NULL);

    // Graph type loop (0, 1)
    for(int i=0; i<m_iGraphTypeNum; i++)
    {
        // set graph structure
        setGraph(m_pGraphTypeArray[i]);

        // if the graph type is tree
```

```
            if ((m_pGraphTypeArray[i] == 1) && (m_bAggregation))
            {
                // do aggregation related tasks
                setupAggregationLists();
            }

            // if m_bAggregation is true then do experiments twice otherwise once
            int executionTimes = m_bAggregation? 2 : 1;

            // reset the bAggregation flag
            bAggregation = !m_bAggregation;

            // aggregation related experiment
            for (int x=0; x<executionTimes; x++)
            {
                // switch
                bAggregation = !bAggregation;


                // just for debugging
                //if (!bAggregation) continue;

                // if it is a graph type and aggregation flag is true then skip it
                if (bAggregation && (m_pGraphTypeArray[i] == 0)) continue;

                m_players[0].s_bAggregation = bAggregation;

                for(int j=0; j<m_iAlgorithmNum; j++)
                {
                    // set the packet generation algorithm
                    m_players[0].s_iPacketGenerationAlgorithm =
                        m_pAlgorithmArray[j];

                    for(int k=0; k<m_iPlayoutDelayNum; k++)
                    {
                        // set the playout delay for each node
                        m_players[0].s_iPlayoutDelay = m_pPlayoutDelayArray[k];

                        for (int l=0; l<m_iPacketDropRateNum; l++)
                        {
                            doItNow(h, i, j, k, l, strDirectory, bAggregation);
                        }
                    }
                }
            }
        }
    }

    infile.close();
}
```

```
bool GraphToTreeSimulator::createOutputFiles(char *strParameters, char
*strDataDirectory)
{
    char strFileName[MAX_CHAR];

    // create a data directory
    ::CreateDirectory(strDataDirectory, NULL);

    // open outfile for all nodes
    sprintf( strFileName, "%s\\%s", strDataDirectory,
"output_packets_from_all.txt");

    m_outfile[m_iTotalNodeNum].open(strFileName, ios_base::out | ios_base::trunc);

    m_outfile[m_iTotalNodeNum] << setiosflags( ios::left );
    m_outfile[m_iTotalNodeNum] << packetType::showHeader(WIDTH) << endl;

    for (int i=0; i< m_iTotalNodeNum; i++)
    {
       sprintf( strFileName, "%s\\%s%d%s", strDataDirectory, "output_packets_from_",
i, ".txt");
        m_outfile[i].open(strFileName, ios_base::out | ios_base::trunc);

        if (!m_outfile[i])
        {
            return false;
        }

        m_outfile[i] << setiosflags( ios::left );
        m_outfile[i] << packetType::showHeader(WIDTH) << endl;
    }

    return true;
}

void GraphToTreeSimulator::setGraph(int iGraphType)
{
    int iLength;                                    // the length of pathList
    int *pathList = new int[m_iTotalNodeNum];       // a temporary path list
    assert(pathList);

    //  For each vertex v
    for (int i=0; i < m_iTotalNodeNum; i++)
    {
        if (iGraphType == 0)
        {
            // get direct links
            m_graph.directPath(i);
        }
        else
        {
            // Find the shortest path to all adjacent nodes of v
            m_graph.shortestPath(i);
```

```
        }

        // for testing
        // show the shortest path and weight
        m_graph.printShortestDistance(i);
        m_graph.printShortestPath(i);

        for (int j = 0; j < m_iTotalNodeNum; j++)
        {
            m_graph.getShortestPath(i, j, pathList, iLength);

            // Set playerNode
            m_players[i].setPath(j, pathList, iLength);
        }

        m_players[i].setWeights(m_graph.getWeightsFrom(i));
    }

    // for testing
    // printAllPlayersPath();

    delete [] pathList;
}

void GraphToTreeSimulator::setupAggregationLists()
{
    // set aggregation lists
    // pass other node's paths to all nodes
    passAllPaths();

    // set all lists related to aggregation process
    for (int i = 0; i < m_iTotalNodeNum; i++)
    {
        m_players[i].setAggregation();
    }
}

void GraphToTreeSimulator::passAllPaths()
{
    // exchange players' path information to each other
    for(int myPlayerNum = 0; myPlayerNum < m_iTotalNodeNum; myPlayerNum++)
    {
        for (int otherPlayerNum = 0; otherPlayerNum < m_iTotalNodeNum;
otherPlayerNum++)
        {
            // if it is my path information then skip it
            if (myPlayerNum == otherPlayerNum) continue;

            // send other node's path information to the player
            m_players[myPlayerNum].setOtherPath(
                m_players[otherPlayerNum].getAllPath(), otherPlayerNum);
        }
    }
```

```
}

bool GraphToTreeSimulator::readTotalNodeNum(ifstream& infile)
{
    string      strFileName;
    string      tempStr;

    // open the file
    infile.open(m_strDataFileName.data());

    // if the file is not available then go back
    if(!infile)
    {
        cout  << "Can not open the file." << endl;

        return false;
    }

    // get the number of nodes
    while(infile)
    {
        infile >> tempStr;

        // if the file contains "#Nodes: 000"
        if (stricmp(tempStr.data(), "#Nodes:") == 0)
        {
            infile >> m_iTotalNodeNum;

            break;
        }
    }

    if (m_iTotalNodeNum == 0)
    {
        cout << "Can not find total node number." << endl;

        return false;
    }

    return true;
}

bool GraphToTreeSimulator::startSimulation(int iPacketDropRate)
{
    priorityQueueType<packetType> mainBuffer;   // Set the main buffer structure

    int iMeasureTime = 1000 * 1;    // 1 seconds
    int iSourceNodeTemp;
    int iDest;
    bool bDropped = true;

    packetType pacPacket;           // packet variable
```

```
    // Start simulation loop
    // The value 'i' will be the step of milli-second each
    for (int i=1; i<=m_iSimulationTime * 1000; i++)
    {
        // show a message every 1 second passes
        if (i % iMeasureTime == 0)
        {
            cout << (i / iMeasureTime) << " seconds passed." << endl;
        }

        // if the player generated a packet then add it into main buffer
        for (int j=0; j<m_iTotalNodeNum; j++)
        {
            // send simulation time
            m_players[j].setDepartureTime(i);

            m_players[j].generatePacket(mainBuffer);
        }

        // check main buffer whether it contains packets to send
        while (!mainBuffer.isEmpty())
        {
            // peek it
            pacPacket = mainBuffer.front();

            // if the front of main buffer contains packets which has future
            // arrival time then get out
            if (pacPacket.getArrivalTime() > i) break;

            // get the source node
            iSourceNodeTemp = pacPacket.getSourceNode();


            // get origin player node
            // get the first destination player node
            iDest = pacPacket.getDestination().front();


            // get drop rate & if the packet is not dropped then do packet relay
            if ( (rand() % 100) >= iPacketDropRate)
            {
                // set bDropped to false - it is not a dropped packet
                bDropped = false;

                m_players[iDest].addPacket(pacPacket);


                // for statistics
                // store packets to all nodes file
                m_outfile[m_iTotalNodeNum] << setw(WIDTH) << "0" << pacPacket <<
endl;

                // store packets to the source node file
```

```
                m_outfile[iSourceNodeTemp] << setw(WIDTH) << "0" << pacPacket <<
endl;
            }
            else
            {
                // set bDropped to true - it is a dropped packet
                bDropped = true;


                // for statistics
                // store dropped packets to all nodes file
                m_outfile[m_iTotalNodeNum] << setw(WIDTH) << "1" << pacPacket <<
endl;

                // store dropped packets to the source node file
                m_outfile[iSourceNodeTemp] << setw(WIDTH) << "1" << pacPacket <<
endl;
            }

            mainBuffer.deleteQueue();


            // For statistics
            m_statistics.addPacket(pacPacket, bDropped);
        }

        for (int j=0; j<m_iTotalNodeNum; j++)
        {
            // each payer process arrived packets
            m_players[j].processPackets(mainBuffer);

            // re-send dropped packets
            m_players[j].resendPackets(mainBuffer);
        }

        // check key input
        if (kbhit())
        {
            getch();        // throw key away

            cout << "Simulation is interrupted." << endl;

            return false;
        }
    }

    return true;
}

void GraphToTreeSimulator::cleanUpAll()
{
    for (int i=0; i<m_iTotalNodeNum; i++)
    {
```

```
        m_players[i].cleanUpAll();
    }

    m_statistics.cleanUpAll();
}

void GraphToTreeSimulator::printAllPlayersPath()
{
    int *pathList = new int[m_iTotalNodeNum];
    int iLength;

    for (int i = 0; i < m_iTotalNodeNum; i++)
    {
        for (int j = 0; j < m_iTotalNodeNum; j++)
        {
            m_players[i].getPath(j, pathList, iLength);

            for (int k = 0; k < iLength; k++)
            {
                cout << pathList[k] << " ";
            }
            cout << endl;
        }
        cout << endl;
    }

    delete [] pathList;
}
```

statistics.h

```
#include "packetType.h"
#include "nodeInfo.h"

class Statistics
{
public:
    Statistics();

    ~Statistics();

    void setStatistics(int iNodeNum, int iMaxFrame);

    void addPacket(packetType packet, bool bDropped);

    void saveParametersIntoFile(char* strDirName,
            int& iGraphType, int& iSimulationTime, int& iPacketGenerationAlgorithm,
            int& iFrameInterval, int& iPlayoutDelay, int& iPacketDropRate, bool
bAgg);

    void saveLatencyReportIntoFile(char* strDirName);
```

```
    void saveExperimentalResultIntoFile(char* strDirName);

    void cleanUpAll();

    // for statistics
    unsigned long   m_lTotalLatencyForFrameOnly;
    unsigned long   m_lTotalLatencyForAll;

    int             m_iTotalDroppedPackets;
    int             m_iTotalArrivedPackets;

private:

    NodeInfo        *m_NodeInfo;
    int             m_iNodeNum;
};

Statistics::Statistics()
{
    m_NodeInfo = NULL;
}

void Statistics::setStatistics(int iNodeNum, int iMaxFrame)
{
    m_iTotalDroppedPackets = 0;
    m_iTotalArrivedPackets = 0;

    m_lTotalLatencyForAll = 0;
    m_lTotalLatencyForFrameOnly = 0;

    // set max frame of NodeInfo class
    NodeInfo::s_iMaxFrame = iMaxFrame;

    // set node num of Frame class
    Frame::s_iNodeNum = iNodeNum;

    m_NodeInfo = new NodeInfo[iNodeNum];

    // set node id
    for (int i=0; i<iNodeNum; i++)
    {
        m_NodeInfo[i].setNodeID(i);
    }

    m_iNodeNum = iNodeNum;
}

Statistics::~Statistics()
{
    delete [] m_NodeInfo;
}

void Statistics::addPacket(packetType packet, bool bDropped)
```

```
{
    // count dropped & arrived packets
    if (bDropped) m_iTotalDroppedPackets++;
    else m_iTotalArrivedPackets++;

    // if it is not a ACK packet nor a dropped packet then add it
    if ((!bDropped) && (packet.getPacketType() != ACK))
    {
        m_NodeInfo[packet.getSourceNode()].addPacket(packet);
    }

    // calcaulate total latency
    m_lTotalLatencyForAll += packet.getArrivalTime() - packet.getDepartureTime();
}


void Statistics::saveParametersIntoFile(char* strDirName,
            int& iGraphType, int& iSimulationTime, int& iPacketGenerationAlgorithm,
            int& iFrameInterval, int& iPlayoutDelay, int& iPacketDropRate, bool
bAgg)
{
    ofstream parameterFile;
    char parameterFileName[256];

    sprintf( parameterFileName, "%s\\%s", strDirName, "Parameter_report.txt" );

    parameterFile.open(parameterFileName, ios_base::out | ios_base::trunc);

    parameterFile << "#iGraphType (0: Graph, 1: Tree): " << iGraphType << endl;
    parameterFile << "#iSimulationTime (in seconds): " << iSimulationTime << endl;
    parameterFile << "#iPacketGenerationAlgorithm " <<
        "(0: frequent packet regeneration, 1: lock-step, 2: bucket-synch): " <<
        iPacketGenerationAlgorithm << endl;
    parameterFile << "#iFrameInterval (in milliseconds): " << iFrameInterval <<
endl;
    parameterFile << "#iPlayoutDelay (in milliseconds): " << iPlayoutDelay << endl;
    parameterFile << "#iPacketDropRate (whole number %): " << iPacketDropRate <<
endl;
    parameterFile << "#bAggregation (true: 1): " << bAgg << endl;

    parameterFile.close();
}

void Statistics::saveLatencyReportIntoFile(char* strDirName)
{
    ofstream reportFile;
    char reportFileName[256];

    for (int i=0; i<m_iNodeNum; i++)
    {
       sprintf( reportFileName, "%s\\%s%d%s", strDirName, "Latency_report_node_", i,
".txt" );
        reportFile.open(reportFileName, ios_base::out | ios_base::trunc);
```

```
        // print header
        reportFile << setiosflags( ios::left );
        reportFile << setw(8) << "#NODE" << setw(8) << "#FRAME" <<
            setw(8) << "#T_L" << setw(8) << "#A_L" <<
            setw(8) << "#M_L" << setw(8) << "#T_L" << endl;

        // report latency
        m_NodeInfo[i].showLatency(reportFile);

        reportFile.close();
    }
}

void Statistics::saveExperimentalResultIntoFile(char* strDirName)
{
    ofstream    resultFile;
    char        resultFileName[256];
    int         iTotalPackets;
    int         iTotalFramePackets = 0;

    sprintf( resultFileName, "%s\\%s", strDirName, "Result_report.txt" );

    resultFile.open(resultFileName, ios_base::out | ios_base::trunc);

    // save result
    resultFile << setiosflags( ios::left );
    resultFile << "#Experimental result report from the statistics class." << endl;

    // save bandwidth result
    iTotalPackets = m_iTotalDroppedPackets + m_iTotalArrivedPackets;
    resultFile << "#Total dropped packets: " << m_iTotalDroppedPackets << endl;
    resultFile << "#Total arrived packets: " << m_iTotalArrivedPackets << endl;
    resultFile << "#Total packets: " << iTotalPackets << endl;

    // save latency result
    for (int i=0; i<m_iNodeNum; i++)
    {
        m_lTotalLatencyForFrameOnly += m_NodeInfo[i].getTotalLatency();
    }

    resultFile << "#Total latency for all: " << m_lTotalLatencyForAll << endl;
    resultFile << "#Total latency for frame only: " << m_lTotalLatencyForFrameOnly
<< endl;

    // save the result of average latency for all
    resultFile << "#Average latency for all: " <<
        (double)m_lTotalLatencyForAll / iTotalPackets << endl;

    // save the result of average latency for frame only
    for (int i=0; i<m_iNodeNum; i++)
    {
        iTotalFramePackets += m_NodeInfo[i].getTotalFramePackets();
```

```
    }

    resultFile << "#Average latency for frame only: " <<
        (double)m_lTotalLatencyForFrameOnly / iTotalFramePackets << endl;


    // save the maximum frame number
    for (int i=0; i<m_iNodeNum; i++)
    {
        resultFile << "#Max Frame (" << i << "): " << m_NodeInfo[i].getMaxFrame()
<< endl;
    }


    resultFile.close();
}

void Statistics::cleanUpAll()
{
    for (int i=0; i<m_iNodeNum; i++)
    {
        m_NodeInfo[i].cleanUpAll();
    }

    // init member variables
    m_lTotalLatencyForFrameOnly = 0;
    m_lTotalLatencyForAll = 0;

    m_iTotalDroppedPackets = 0;
    m_iTotalArrivedPackets = 0;
}
```

Appendix B

Duel-X (Game Module)

m_CPlayer.h

```
#pragma once
#pragma warning( disable: 4251 )

#include <vector>

#include "Common.h"
#include "m_GameData.h"
#include "..\\Library\\WinSockDLL\\m_CSession.h"

#include "DataStructure\\weightedGraphType.h"
#include "DataStructure\\priorityQueueType.h"
#include "DataStructure\\circularArray.h"

// we are using circular array, so we only need a few frames such as 10 to 20 but
// for fair analysis, it is set for 2500 frames (100 seconds)
const int MAX_FRAME_NUM_FOR_GAME_BUFFER = 2500;


#define MAX_SHIP_X      (MAX_SCREEN_X - 32)
#define MAX_SHIP_Y      (MAX_SCREEN_Y - 32)
#define MAX_SHIP_FRAME 40
#define MAX_BULLET_X     (MAX_SCREEN_X - 3)
#define MAX_BULLET_Y     (MAX_SCREEN_Y - 3)
#define MAX_BULLET_FRAME 400

#define NUM_SHIP_TYPES 4

struct SHIP
{
    double              dPosX, dPosY;               // ship x and y position
    double              dBulletPosX, dBulletPosY;   // bullet x and y position
    DWORD               dwBulletFrame;              // bullet frame
    char                cFrame;                     // current ship frame
    BYTE                byType;                     // ship type
    BOOL                bEnable;                    // is this ship active?
    BOOL                bBulletEnable;              // Is there an active bullet?

    // ship x and y velocity (pixels/millisecond)
    double              dVelX, dVelY;

    // bullet x and y velocity (pixels/millisecond)
    double              dBulletVelX, dBulletVelY;
```

```
    DWORD               dwScore;        // current score
//    DWORD               dwLastTick;   // most recent time stamp

    BOOL                bIgnore;        // flag used to synchronize ship explosions
    int                 iCountDown;     // enable time-out
    DWORD               dwFrameCount;   // number of frames since beginning of time

};


enum m_PLAYER_TYPE {MYSELF = 0, LOBBY, PLAYERS};


class m_CPlayer
{
private:
    m_CSession      m_Session;

    string      m_strID;
    int         m_iPlayerNum;
    int         m_iGroupNo;
    int         m_iType;
    int         m_iEnergy;

    unsigned int    m_uiRenderFrameNo;  // the frame no that we will see
    unsigned int    m_uiPlayFrameNo;    // the frame no that we are playing now

    unsigned int    m_uiPacketID;       // unique packet id

public:
    SHIP        m_ship;

    m_CPlayer(void);
    ~m_CPlayer(void);

    // initialize our ship data
    void InitShip();

    unsigned int    GetPacketID() { return m_uiPacketID; }
    unsigned int    GetUniquePacketID() { return m_uiPacketID++; }

    void    SetType(int iType) {m_iType = iType;}
    int     GetType() {return m_iType;}

    void    SetID(string ID) {m_strID = ID;}
    string  GetID() {return m_strID;}

    void    SetPlayerNum(int playerNum) {m_iPlayerNum = playerNum;}
    int     GetPlayerNum() {return m_iPlayerNum;}

    unsigned int    GetPlayFrameNo() {return m_uiPlayFrameNo;}
    void            SetPlayFrameNo(unsigned int frameNo) {m_uiPlayFrameNo =
frameNo;}
```

```
    void            IncreasePlayFrameNo();

    void            SetRenderFrameNo(unsigned int frameNo)
                    {m_uiRenderFrameNo = frameNo;}
    unsigned int    GetRenderFrameNo() {return m_uiRenderFrameNo;}
    void            IncreaseRenderFrameNo();


    SOCKET  GetKey() const { return m_Session.GetTCP().GetSocket(); }
    void    SetKey(SOCKET socket) { m_Session.SetSocket(socket);}

    m_CSession&     GetSession() {return m_Session;}

   bool operator == (const m_CPlayer &player) {return GetKey() == player.GetKey();}
    bool operator != (const m_CPlayer &player)
                    {return GetKey() != player.GetKey();}
    bool operator > (const m_CPlayer &player)  {return GetKey() > player.GetKey();}
    bool operator < (const m_CPlayer &player)  {return GetKey() < player.GetKey();}

    // For Debug
    void ShowPath();
    void ShowWeights();

    int randInt( int low, int high );



    // For Tree-based P2P System
    void setPath(int toNode, int pathList[], int iLength);
    void setWeights(double weights[]);


    // Save RTT values from other players
    void SaveRTT(double RTT_Array[M_iTOTAL_PLAYERS][M_iTOTAL_PLAYERS], int
iNodeNum);

    // Set the graph & tree structure
    void SetGraph(int iGraphType, vector <int>& vPlayers);

    void SaveGameData(int iPacketType, BYTE* strData, int iSize, int iSender,
        ExperimentInfo eInfo, bool bLocal);

    void SaveForStatistics(int iPacketType, BYTE* strData, int iSize, int iSender,
        bool bInbound);

    // Is time for next frame?
    BOOL IsTimeForPacketGeneration(ExperimentInfo eInfo, int iTotalPlayers);
    BOOL IsTimeForUpdate(ExperimentInfo eInfo, int iTotalPlayers);

    // Set my ship data
    void SetShip(stPacketStatusGraph packet);

    // Read GameBuffer
```

```
    BOOL GetFrameData(int iFrameNo, vector<m_GameData>& gameDataVector);
    // Delete game data for the frame from the game buffer
    void DeleteGameData(int iFrameNo);

    // Add the packet into resending queue
    void PutIntoResendingBuffer(int iPacketType, BYTE* strData, int iSize, int
iReceiver);

    void ResendPackets();

    // Process ACK
    void ProcessACK(int iPacketType, BYTE* strData, int iSize, int iSender);

    void GetReceiverVector(int iDest, vector<int>& iPathVector);

    void ApplyGameData(m_GameData gameData, int iFrameInterval);

    void FSR_ApplyGameData(stPacketStatusGraph packet);

protected:
    weightedGraphType<int, M_iTOTAL_PLAYERS> m_graph;
    vector<int> m_path[M_iTOTAL_PLAYERS];       // keep its own paths to other
nodes
    double m_dWeights[M_iTOTAL_PLAYERS];

    priorityQueueType<m_GameData> m_resendingQueue;
    circularArray<m_GameData, MAX_FRAME_NUM_FOR_GAME_BUFFER> m_gameBuffer;

};
```

m_GameData.h

```
#ifndef __m_GameData
#define __m_GameData


#include <string>
#include <vector>

#include "Common.h"

using namespace std;

class m_GameData
{
private:

public:
    // Packet Data type (Status, Commend ... )
    int             m_iPacketType;

    // Transmit type (INIT, RESEND, ACK ...)
```

```
int             m_iTransmitType;

int             m_iOriginalSender;          // original sender id
int             m_iReceiver;                // receiver id for resending

BYTE            m_strData[100];             // packet data

// This packet departed at this time
unsigned int    m_iDepartedTime;

// This packet must depart after this time
unsigned int    m_iWillDepartTime;

unsigned int    m_iReceiverList;            // receiver id list in bits form

int             m_iFrameNo;                 // Frame number

// How many times did we re-send this packet
int             m_iResendFrequency;

m_GameData()
{
    m_iPacketType = iUnknown;
    m_iTransmitType = iUnknown;
    m_iOriginalSender = iUnknown;
    m_iReceiver = iUnknown;

    m_iReceiverList = 0;

    m_iDepartedTime = iUnknown;
    m_iWillDepartTime = iUnknown;

    m_iFrameNo = iUnknown;
    m_iResendFrequency = 0;
}

m_GameData& operator= (const m_GameData& gameData)
{
    if (this != &gameData)  // avoid self-copy
    {
        m_iPacketType = gameData.m_iPacketType;
        m_iTransmitType = gameData.m_iTransmitType;
        m_iOriginalSender = gameData.m_iOriginalSender;

        m_iReceiver = gameData.m_iReceiver;
        m_iReceiverList = gameData.m_iReceiverList;

        m_iDepartedTime = gameData.m_iDepartedTime;
        m_iWillDepartTime = gameData.m_iWillDepartTime;

        m_iFrameNo = gameData.m_iFrameNo;
        m_iResendFrequency = gameData.m_iResendFrequency;
```

```
            memcpy(m_strData, gameData.m_strData, sizeof(m_strData));
        }

        return *this;
    }

    int  GetPacketType() { return m_iPacketType; }
    void SetPacketType(int iType) { m_iPacketType = iType; }

    int  GetTransmitType() { return m_iTransmitType; }
    void SetTransmitType(int iType) { m_iTransmitType = iType; }

    int  GetOriginalSender() { return m_iOriginalSender; }
    void SetOriginalSender(int iOriginalSender) { m_iOriginalSender =
iOriginalSender; }

    int  GetReceiver() { return m_iReceiver; }
    void SetReceiver(int iReceiver) { m_iReceiver = iReceiver; }

    BYTE* GetStrData() { return m_strData; }
    void  SetStrData(stPacketCommandGraph packet){memcpy(m_strData, &(packet),
sizeof(packet));}
    void  SetStrData(stPacketStatusGraph packet){memcpy(m_strData, &(packet),
sizeof(packet));}
   void  SetStrData(BYTE* strData, int iSize){ memcpy(m_strData, strData, iSize); }

    unsigned int    GetReceiverList() { return m_iReceiverList; }
    void            SetReceiverList(unsigned int uiReceiverList)
    {
        m_iReceiverList = uiReceiverList;
    }

    unsigned int    GetDepartedTime() { return m_iDepartedTime; }
    void           SetDepartedTime(unsigned int iTime) { m_iDepartedTime = iTime; }

    unsigned int    GetWillDepartTime() { return m_iWillDepartTime; }
    void            SetWillDepartTime(unsigned int iTime) { m_iWillDepartTime =
iTime; }
    void            AddWillDepartTime(unsigned int iTime) { m_iWillDepartTime +=
iTime; }

    int   GetFrameNo() { return m_iFrameNo; }
    void  SetFrameNo(int iFrameNo) { m_iFrameNo = iFrameNo; }

    int   GetResendFrequency() { return m_iResendFrequency; }
    void  SetResendFrequency(int iFrequency) { m_iResendFrequency = iFrequency; }
    void  IncreaseResendFrequency() { m_iResendFrequency++; }

    bool operator==(const m_GameData& gameData)
    {
        return ( (m_iOriginalSender == gameData.m_iOriginalSender) &&
            (m_iFrameNo == gameData.m_iFrameNo) );
    }
```

```
    bool operator^(const m_GameData& gameData)
    {
        return ( m_iOriginalSender == gameData.m_iOriginalSender );
    }

    bool operator>=(const m_GameData& gameData)
    {
        return (m_iWillDepartTime >= gameData.m_iWillDepartTime);
    }

    bool operator>(const m_GameData& gameData)
    {
        return (m_iWillDepartTime > gameData.m_iWillDepartTime);
    }

    bool operator<=(const m_GameData& gameData)
    {
        return (m_iWillDepartTime <= gameData.m_iWillDepartTime);
    }

    // this is for sort ...
    bool operator<(const m_GameData& gameData)
    {
        return (m_iOriginalSender <= gameData.m_iOriginalSender);
    }


};

#endif
```

Appendix C

Duel-X (Lobby Module)

DuelX_LobbyDoc.h

```
#pragma once

// DuelX_LobbyDoc.h : interface of the CDuelX_LobbyDoc class
//

#ifndef _LOBBYDOC_H
#define _LOBBYDOC_H

#include ".¥WinSock¥m_CGameRoom.h"
#include ".¥WinSock¥m_CPlayer.h"
#include ".¥WinSock¥linkedlist.h"
#include "Users.h"

class m_CSession;

class CDuelX_LobbyDoc : public CDocument
{
protected: // create from serialization only
    CDuelX_LobbyDoc();
    DECLARE_DYNCREATE(CDuelX_LobbyDoc)

// Attributes
public:
    CUsers              m_UsersSet;
    CSession            m_Session;

    // AVL Tree for keeping connected TCP Socket List
    // m_CSession       m_SessionManager;
    //  AVL <m_CPlayer, string>             m_PlayerList;
    //  AVL <m_CGameRoom, string>           m_GameRoomList;

    linkedListType <m_CPlayer>          m_PlayerList;
    linkedListType <m_CGameRoom>        m_GameRoomList;

    m_CPlayer       m_PlayerMyself;

    CDBPropSet      m_Prop;



    // Operations
public:
```

```
// Overrides
    public:
    virtual BOOL OnNewDocument();
    virtual void Serialize(CArchive& ar);

// Implementation
public:
    virtual ~CDuelX_LobbyDoc();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
    DECLARE_MESSAGE_MAP()
public:
    virtual void OnCloseDocument();

    bool OnStartServer(int nPort);
    bool OnAsync(UINT wParam, LPARAM lParam);
    void     SendAuthResult(m_CPlayer *, bool bResult);
    void     SendCreateGameRoomResult(m_CPlayer *, bool bResult);

    // Prepare game room list packet
    void PrepareGameRoomListPacket(m_CGameRoom &gameRoom, stPacketRoomList
&listPacket);
    // Send game room list
    void SendGameRoomList(m_CPlayer *player, stPacketRoomList *listPacket, int
roomCount);
};

#endif
```

m_CGameRoom.h

```
#pragma once
#include <string>

using namespace std;

class m_CGameRoom
{
private:
    string          m_GameName;
    SOCKET          m_Socket;       // unique key
    string          m_Password;
    string          m_ID;
    unsigned long   m_IP;
    unsigned short  m_Port;
```

```
public:
    m_CGameRoom(void);
    ~m_CGameRoom(void);

    SOCKET  GetKey() const {return m_Socket;}

    string      GetGameName() {return m_GameName;}
    void        SetGameName(string gameName) { m_GameName = gameName;}

    SOCKET      GetSocket() {return m_Socket;}
    void        SetSocket(SOCKET socket) { m_Socket = socket;}

    string      GetPassword() {return m_Password;}
    void        SetPassword(string password) { m_Password = password;}

    string      GetID() {return m_ID;}
    void        SetID(string ID) { m_ID = ID;}

    unsigned long   GetIP() {return m_IP;}
    void            SetIP(unsigned long ip) {m_IP = ip;}

    unsigned short  GetPort() {return m_Port;}
    void            SetPort(unsigned short port) {m_Port = port;}

    void SetAll(char *gameName, char *password, unsigned long  ip, unsigned short
port, string ID)
    {
        SetGameName(string(gameName));
        SetPassword(string(password));

        SetIP(ip);
        SetPort(port);

        SetID(ID);
    }

    bool operator == (const m_CGameRoom& gameRoom) {return GetKey() ==
gameRoom.GetKey();}
    bool operator != (const m_CGameRoom& gameRoom)  {return GetKey() ==
gameRoom.GetKey();}
    bool operator > (const m_CGameRoom& gameRoom)  {return GetKey() ==
gameRoom.GetKey();}
    bool operator < (const m_CGameRoom& gameRoom)  {return GetKey() ==
gameRoom.GetKey();}

};
```

Appendix D

Duel-X (MDX-Reader: Analysis Tool)

MDX_ReaderDlg.cpp

```
// MDX_ReaderDlg.cpp : implementation file

#include "stdafx.h"
#include "MDX_Reader.h"
#include "MDX_ReaderDlg.h"
#include "RTTDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CMDX_ReaderDlg dialog

CMDX_ReaderDlg::CMDX_ReaderDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CMDX_ReaderDlg::IDD, pParent)
    , m_uiStartTime(0)
    , m_uiFinishTime(0)
    , m_uiElapsedTime(0)
    , m_uiLastFrame(0)
    , m_dFPS(0)
    , m_strP2PType(_T(""))
    , m_strSyncAlgorithm(_T(""))
    , m_strTransmission(_T(""))
    , m_strAggregation(_T(""))
    , m_iFrameInterval(0)
    , m_iPlayoutDelay(0)
    , m_iMyPlayerNum(0)
    , m_iTotalPlayers(0)
    , m_iTotalPackets(0)
    , m_iInboundTotal(0)
    , m_iInboundACK(0)
    , m_iInboundGame(0)
    , m_iInboundResend(0)
    , m_iOutboundTotal(0)
    , m_iOutboundACK(0)
    , m_iOutboundGame(0)
    , m_iOutboundResend(0)
    , m_strOpenFilter(_T(""))
    , m_strOpenValue(_T(""))
    , m_strViewFilter(_T(""))
    , m_iMyLastFrame(0)
    , m_dInPPS(0)
```

```
    , m_dOutPPS(0)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CMDX_ReaderDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_LIST_DATA, m_DataList);

    m_iCounter = 0;
    DDX_Text(pDX, IDC_EDIT_START_TIME, m_uiStartTime);
    DDX_Text(pDX, IDC_EDIT_FINISH_TIME, m_uiFinishTime);
    DDX_Text(pDX, IDC_EDIT_ELAPSED_TIME, m_uiElapsedTime);
    DDX_Text(pDX, IDC_EDIT_LAST_FRAME, m_uiLastFrame);
    DDX_Text(pDX, IDC_EDIT_FPS, m_dFPS);
    DDX_Text(pDX, IDC_EDIT_P2P_TYPE, m_strP2PType);
    DDX_Text(pDX, IDC_EDIT_SYNC_ALG, m_strSyncAlgorithm);
    DDX_Text(pDX, IDC_EDIT_TRANSMISSION, m_strTransmission);
    DDX_Text(pDX, IDC_EDIT_AGGREGATION, m_strAggregation);
    DDX_Text(pDX, IDC_EDIT_FRAME_INTERVAL, m_iFrameInterval);
    DDX_Text(pDX, IDC_EDIT_PLAYOUT_DELAY, m_iPlayoutDelay);
    DDX_Text(pDX, IDC_EDIT_MY_PLAYER_NUM, m_iMyPlayerNum);
    DDX_Text(pDX, IDC_EDIT_TOTAL_PLAYERS, m_iTotalPlayers);
    DDX_Text(pDX, IDC_EDIT_TOTAL_PACKETS, m_iTotalPackets);
    DDX_Text(pDX, IDC_EDIT_INBOUND_TOTAL, m_iInboundTotal);
    DDX_Text(pDX, IDC_EDIT_IN_ACK, m_iInboundACK);
    DDX_Text(pDX, IDC_EDIT_IN_GAME, m_iInboundGame);
    DDX_Text(pDX, IDC_EDIT_IN_RESEND, m_iInboundResend);
    DDX_Text(pDX, IDC_EDIT_OUT_TOTAL, m_iOutboundTotal);
    DDX_Text(pDX, IDC_EDIT_OUT_ACK, m_iOutboundACK);
    DDX_Text(pDX, IDC_EDIT_OUT_GAME, m_iOutboundGame);
    DDX_Text(pDX, IDC_EDIT_OUT_RESEND, m_iOutboundResend);
    DDX_Control(pDX, IDC_COMBO_OPEN_FILTER, m_ctrlOpenFilter);
    DDX_CBString(pDX, IDC_COMBO_OPEN_FILTER, m_strOpenFilter);
    DDX_Text(pDX, IDC_EDIT_OPEN_VALUE, m_strOpenValue);
    DDX_Control(pDX, IDC_COMBO_FILTER, m_ctrlViewFilter);
    DDX_CBString(pDX, IDC_COMBO_FILTER, m_strViewFilter);
    DDX_Text(pDX, IDC_EDIT_MY_LAST_FRAME, m_iMyLastFrame);
    DDX_Text(pDX, IDC_EDIT_PPS_IN, m_dInPPS);
    DDX_Text(pDX, IDC_EDIT_PPS_OUT, m_dOutPPS);
}

BEGIN_MESSAGE_MAP(CMDX_ReaderDlg, CDialog)
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //}}AFX_MSG_MAP
    ON_BN_CLICKED(IDC_BUTTON_OPEN, OnBnClickedButtonOpen)
    ON_BN_CLICKED(IDC_BUTTON_CLEAR, OnBnClickedButtonClear)
    ON_BN_CLICKED(IDC_BUTTON_RTT_FRAME, OnBnClickedButtonRttFrame)
    ON_BN_CLICKED(IDC_BUTTON_SAVE, OnBnClickedButtonSave)
    ON_BN_CLICKED(IDC_BUTTON_DELETE_ITEMS, OnBnClickedButtonDeleteItems)
    ON_BN_CLICKED(IDC_BUTTON_FRAME_INTERVAL, OnBnClickedButtonFrameInterval)
```

```
END_MESSAGE_MAP()


// CMDX_ReaderDlg message handlers

BOOL CMDX_ReaderDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Set the icon for this dialog.  The framework does this automatically
    //  when the application's main window is not a dialog
    SetIcon(m_hIcon, TRUE);         // Set big icon
    SetIcon(m_hIcon, FALSE);        // Set small icon

    // TODO: Add extra initialization here

    // Init Data List Control Columns
    InitGameDataColumn();

    // Init Open Filter Control
    for (int i=0; i < g_iOpenFilterSize; i++)
    {
        m_ctrlOpenFilter.AddString(g_strOpenFilter[i]);
        m_ctrlViewFilter.AddString(g_strOpenFilter[i]);
    }

    return TRUE;  // return TRUE  unless you set the focus to a control
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CMDX_ReaderDlg::InitData()
{
    m_iCounter = 0;

    m_uiStartTime = 0;
    m_uiFinishTime = 0;
    m_uiElapsedTime = 0;
    m_uiLastFrame = 0;
    m_dFPS = 0;

    m_iTotalPackets = 0;

    m_iInboundTotal = 0;
    m_iInboundACK = 0;
    m_iInboundGame = 0;
    m_iInboundResend = 0;

    m_iOutboundTotal = 0;
    m_iOutboundACK = 0;
    m_iOutboundGame = 0;
```

```
    m_iOutboundResend = 0;

    m_iOpenValueFirst = 0;
    m_iOpenValueLast = 0;
    m_strOpenValueFirst = "";
    m_strOpenValueLast = "";
    m_bIsOpenValueEmpty = TRUE;

    m_iMyLastFrame = 0;


    UpdateData(FALSE);      // update the values of controls
}

void CMDX_ReaderDlg::InitGameDataColumn()
{
    DeleteAllColumns();

    CRect rect;
    m_DataList.GetClientRect(&rect);

    m_DataList.InsertColumn(0, g_strOpenFilter[0], LVCFMT_LEFT, 60);
    m_DataList.InsertColumn(1, g_strOpenFilter[1], LVCFMT_LEFT, 90);
    m_DataList.InsertColumn(2, g_strOpenFilter[2], LVCFMT_LEFT, 60);
    m_DataList.InsertColumn(3, g_strOpenFilter[3], LVCFMT_LEFT, 80);
    m_DataList.InsertColumn(4, g_strOpenFilter[4], LVCFMT_LEFT, 80);
    m_DataList.InsertColumn(5, g_strOpenFilter[5], LVCFMT_LEFT, 80);
    m_DataList.InsertColumn(6, g_strOpenFilter[6], LVCFMT_LEFT, 90);

    // m_DataList.InsertColumn(7, _T("Will Depart Time"), LVCFMT_LEFT, rect.Width()
- 540);
    m_DataList.InsertColumn(7, g_strOpenFilter[7], LVCFMT_LEFT, 80);

    m_DataList.SetExtendedStyle(LVS_EX_FULLROWSELECT | LVS_EX_GRIDLINES);
}

void CMDX_ReaderDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, reinterpret_cast<WPARAM>(dc.GetSafeHdc()),
0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;
```

```
        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialog::OnPaint();
    }
}

// The system calls this function to obtain the cursor to display while the user
drags
//  the minimized window.
HCURSOR CMDX_ReaderDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

void CMDX_ReaderDlg::OnBnClickedButtonOpen()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);   // save data from control into variables

    // parse the string in open filter edit control
    if (!ParseFilterString(m_iOpenValueFirst,   m_iOpenValueLast,
m_strOpenValueFirst,
        m_strOpenValueLast, m_strOpenValue, m_bIsOpenValueEmpty)) return;

    // if everything is ok then open a file and read data
    char szFilter[] = "MDX Files (*.mdx)|*.mdx|All Files (*.*)|*.*||";
    CFileDialog dlg(TRUE, NULL, NULL, OFN_HIDEREADONLY, szFilter);

    if (IDOK == dlg.DoModal())
    {
        CFile file;
        file.Open(dlg.GetPathName(), CFile::modeRead);
        CArchive ar(&file, CArchive::load);

        CGameData gameData;

        int iMyPlayerNum, iTotalPlayers;

        TRY
        {
            // Read game info first
            gameData.Serialize(ar, m_expInfo, iMyPlayerNum, iTotalPlayers);

            // Show the game info
            ShowGameInfo(m_expInfo, iMyPlayerNum, iTotalPlayers);

            while(1)
            {
                ar >> gameData;
```

```
                WriteGameDataOnList(gameData);
            }
        }
        CATCH(CArchiveException, e)
        {
            // skip it
            // e->ReportError();
        }
        CATCH(CFileException, e)
        {
            e->ReportError();
        }
        END_CATCH


        // Now show the statistics
        ShowStatistics();
    }
}

void CMDX_ReaderDlg::WriteGameDataOnList(CGameData& gameData)
{
        CString str;
        UINT uiItem[g_iOpenFilterSize];

        UINT    uiLastFrame = gameData.m_gameData.GetFrameNo();
        int     iOriginalSender = gameData.m_gameData.GetOriginalSender();
        int     iReceiver = gameData.m_gameData.GetReceiver();
        UINT    uiReceiverList = gameData.m_gameData.GetReceiverList();
        int     iPacketType = gameData.m_gameData.GetPacketType();
        int     iTransmitType = gameData.m_gameData.GetTransmitType();
        UINT    uiDepartedTime = gameData.m_gameData.GetDepartedTime();
        int     iResendFrequency = gameData.m_gameData.GetResendFrequency();

        // change items into array values
        uiItem[0] = uiLastFrame;
        uiItem[1] = iOriginalSender;
        uiItem[2] = iReceiver;
        uiItem[3] = uiReceiverList;
        uiItem[4] = iPacketType;
        uiItem[5] = iTransmitType;
        uiItem[6] = uiDepartedTime;
        uiItem[7] = iResendFrequency;

        // check open filter combo box
        int iSel = m_ctrlOpenFilter.GetCurSel();

        // if something selected
        if (iSel >= 0)
        {
            // first condition is set
            if (m_strOpenValueFirst != "No")
            {
```

```
                // if first condition is false
                if (!CheckCondition(m_iOpenValueFirst, m_strOpenValueFirst,
uiItem[iSel]))
                {
                    return;
                }
            }
            // second condition is set
            if (m_strOpenValueLast != "No")
            {
                // if second condition is false
                if (!CheckCondition(uiItem[iSel],
                        m_strOpenValueLast, m_iOpenValueLast))
                {
                    return;
                }
            }
        }

        str.Format("%u", uiLastFrame);

        m_DataList.InsertItem(m_iCounter, str, 0);

        str.Format("%i", iOriginalSender);
        m_DataList.SetItemText(m_iCounter, 1, str);

        str.Format("%i", iReceiver);
        m_DataList.SetItemText(m_iCounter, 2, str);

        str.Format("%u", uiReceiverList);
        m_DataList.SetItemText(m_iCounter, 3, str);

        str.Format("%i", iPacketType);
        m_DataList.SetItemText(m_iCounter, 4, str);

        str.Format("%i", iTransmitType);
        m_DataList.SetItemText(m_iCounter, 5, str);

        str.Format("%u", uiDepartedTime);
        m_DataList.SetItemText(m_iCounter, 6, str);

        // str.Format("%u", gameData.m_gameData.GetWillDepartTime());
        // m_DataList.SetItemText(m_iCounter, 7, str);

        str.Format("%i", iResendFrequency);
        m_DataList.SetItemText(m_iCounter, 7, str);

        // calculate Time & Frame related values
        if ( (m_uiStartTime > uiDepartedTime) || (m_iCounter == 0) )
        {
            m_uiStartTime = uiDepartedTime;
        }
```

```
        if (m_uiFinishTime < uiDepartedTime)
        {
            m_uiFinishTime = uiDepartedTime;
        }

        if (m_uiLastFrame < uiLastFrame)
        {
            m_uiLastFrame = uiLastFrame;
        }

        if (iReceiver != m_iMyPlayerNum)
        {
            if (m_iMyLastFrame < (int)uiLastFrame)
            {
                m_iMyLastFrame = (int)uiLastFrame;
            }
        }

        // Calculate Packet related values
        if (iPacketType == PACKET_UDP_ACK)
        {
            if (iOriginalSender != m_iMyPlayerNum)
            {
                m_iInboundACK++;
            }
            else
            {
                m_iOutboundACK++;
            }
        }

        if ( (iPacketType == PACKET_UDP_COMMAND) || (iPacketType ==
PACKET_UDP_STATUS) )
        {
            if (iOriginalSender != m_iMyPlayerNum)
            {
                m_iInboundGame++;
            }
            else
            {
                m_iOutboundGame++;
            }
        }

        // Handle resending packets later
        if ( iResendFrequency > 0 )
        {
            if (iOriginalSender != m_iMyPlayerNum)
            {
                m_iInboundResend++;
            }
            else
            {
```

```
                m_iOutboundResend++;
            }
        }

        m_iCounter++;
}

void CMDX_ReaderDlg::OnBnClickedButtonClear()
{
    // TODO: Add your control notification handler code here
    m_DataList.DeleteAllItemsSuper();

    //m_DataList.DeleteAllItems(ItemdataProc, (LPARAM)this);

    InitData();
}

void CMDX_ReaderDlg::ShowStatistics()
{
//  UpdateData(TRUE);

    m_uiElapsedTime = m_uiFinishTime - m_uiStartTime;

    if (m_uiElapsedTime != 0)
    {
        //m_dFPS = m_uiLastFrame * 1000 / (double)m_uiElapsedTime;

        m_dFPS = m_iMyLastFrame * 1000 / (double)m_uiElapsedTime;

        // set precision to 2
        m_dFPS = (int)(m_dFPS * 100) / 100.0;
    }

    // Set the total number of packets
    m_iTotalPackets = m_iCounter;

    // Set the inbound total packets
    m_iInboundTotal = m_iInboundACK + m_iInboundGame; // + m_iInboundResend;

    // Set the oudbound total packets
    m_iOutboundTotal = + m_iOutboundACK + m_iOutboundGame; // + m_iOutboundResend;

    m_dInPPS = m_iInboundTotal * 1000 / (double)m_uiElapsedTime;
    m_dOutPPS = m_iOutboundTotal * 1000 / (double)m_uiElapsedTime;

    // set precision to 2
    m_dInPPS = (int)(m_dInPPS * 100) / 100.0;
    m_dOutPPS = (int)(m_dOutPPS * 100) / 100.0;

    UpdateData(FALSE);
}

void CMDX_ReaderDlg::ShowGameInfo(ExperimentInfo& expInfo, int iMyPlayerNum,
```

```
                                    int iTotalPlayers)
{
    if (expInfo.iP2P == 0)
    {
        m_strP2PType = _T("Mesh");
    }
    else if (expInfo.iP2P == 1)
    {
        m_strP2PType = _T("Tree");
    }

    if (expInfo.iSynch == 0)
    {
        m_strSyncAlgorithm = _T("Lockstep");
    }
    else if (expInfo.iSynch == 1)
    {
        m_strSyncAlgorithm = _T("LBS");
    }
    else if (expInfo.iSynch == 2)
    {
        m_strSyncAlgorithm = _T("FSR");
    }

    if (expInfo.iTransmission == 0)
    {
        m_strTransmission = _T("None");
    }
    else if (expInfo.iTransmission == 1)
    {
        m_strTransmission = _T("Blind");
    }
    else if (expInfo.iTransmission == 2)
    {
        m_strTransmission = _T("Smart");
    }

    if (expInfo.bAggregation)
    {
        m_strAggregation = _T("Yes");
    }
    else
    {
        m_strAggregation = _T("No");
    }


    m_iFrameInterval = expInfo.iFrameInterval;
    m_iPlayoutDelay = expInfo.iPlayoutDelay;
    m_iMyPlayerNum = iMyPlayerNum;
    m_iTotalPlayers = iTotalPlayers;

    UpdateData(FALSE);
```

```
}

// we assume that the data is sorted on frame number in ascending order
void CMDX_ReaderDlg::OnBnClickedButtonRttFrame()
{
    // TODO: Add your control notification handler code here
    CRTTDialog dlg;
    dlg.DoModal(this);
}

void CMDX_ReaderDlg::DeleteAllColumns()
{
    while (1)
    {
        if (!m_DataList.DeleteColumn(0)) break;
    }
}

void CMDX_ReaderDlg::OnBnClickedButtonSave()
{
    // TODO: Add your control notification handler code here
    UpdateData(TRUE);

    CString strData[g_iOpenFilterSize];

    char szFilter[] = "MDX Files (*.mdx)|*.mdx|All Files (*.*)|*.*||";
    CFileDialog dlg(FALSE, NULL, NULL, OFN_HIDEREADONLY, szFilter);

    if (IDOK == dlg.DoModal())
    {
        TRY
        {
            CFile file;
            file.Open(dlg.GetPathName(), CFile::modeCreate | CFile::modeWrite);
            CArchive ar(&file, CArchive::store);

            // first store game info
            CGameData gameData;
            gameData.Serialize(ar, m_expInfo, m_iMyPlayerNum, m_iTotalPlayers);

            CString str;
            int iLength = m_DataList.GetItemCount();
            int iColumn = m_DataList.GetHeaderCtrl()->GetItemCount();

            // get each row
            for (int i=0; i < iLength; i++)
            {
                // get each column
                for (int j=0; j < iColumn; j++)
                {
                    strData[j] = m_DataList.GetItemText(i, j);
                }
```

```
                // set game data
                gameData.m_gameData.m_iFrameNo = _ttoi(strData[0]);
                gameData.m_gameData.m_iOriginalSender = _ttoi(strData[1]);
                gameData.m_gameData.m_iReceiver = _ttoi(strData[2]);
                gameData.m_gameData.m_iReceiverList = (UINT)(_ttoi(strData[3]));
                gameData.m_gameData.m_iPacketType = _ttoi(strData[4]);
                gameData.m_gameData.m_iTransmitType = _ttoi(strData[5]);
                gameData.m_gameData.m_iDepartedTime = (UINT)(_ttoi(strData[6]));
                gameData.m_gameData.m_iResendFrequency = _ttoi(strData[7]);


                // save data from List Control to our file
                gameData.Serialize(ar);
            }
        }
        CATCH(CFileException, e)
        {
            e->ReportError();
        }
        END_CATCH
    }
}

void CMDX_ReaderDlg::OnBnClickedButtonDeleteItems()
{
    // TODO: Add your control notification handler code here

    POSITION pos;
    while (pos = m_DataList.GetFirstSelectedItemPosition())
    {
        int nSelItem = m_DataList.GetNextSelectedItem(pos);
        m_DataList.DeleteItemSuper(nSelItem);
    }

    // m_DataList.DeleteAllSelectedItems(ItemdataProc, (LPARAM)this);
    //  m_DataList.DeleteAllSelectedItems(ItemdataProc, (LPARAM)this);
    // m_DataList.DeleteItem(m_DataList.GetFirstSelectedItem(), TRUE,
    //  ItemdataProc, (LPARAM)this);
}

BOOL CMDX_ReaderDlg::ItemdataProc(DWORD dwData, LPARAM lParam)
{
    // TODO: Process your item data here

    // Please return TRUE to proceed the deletion, return FALSE to abort.
    return TRUE;
}

BOOL CMDX_ReaderDlg::ParseFilterString(int& iFirstValue, int& iSecondValue,
        CString& strFirstExp, CString& strSecondExp, CString strFilterValue, BOOL&
bEmpty)
{
    // currently minimum data validation is available
```

```
// initialise parameters
iFirstValue = iSecondValue = 0;
strFirstExp = "";
strSecondExp = "";

// if open filter expression is empty then return now
if (strFilterValue.GetLength() == 0)
{
    bEmpty = TRUE;

    return TRUE;
}
else
{
    bEmpty = FALSE;
}

CString strTokens[5];
char c;
int iPosX, iPosFirstValue, iPosSecondValue;


// remove all white space (currently it removes only space)
strFilterValue.Remove(' ');

// insert tokens before and after 'x'

// find char x
iPosX = strFilterValue.Find('x');

// if not found then return
// otherwise, insert tokens before and after 'x'
if (iPosX < 0) return FALSE;
else
{
    // after
    strFilterValue.Insert(iPosX+1, '#');

    // before
    strFilterValue.Insert(iPosX, '#');
}

// insert token before and after the first number value

// find the end of the first number
for (int i=0; i < strFilterValue.GetLength(); i++)
{
    c = strFilterValue.GetAt(i);
    iPosFirstValue = i;

    if (!IsNumber(c)) break;
}
```

```
// if there is no first value then we do not check the first condition
if (iPosFirstValue >= iPosX) strFirstExp = _T("No");
else
{
    // after
    strFilterValue.Insert(iPosFirstValue, '#');

    // before
    strFilterValue.Insert(0, '#');
}

// insert token before and after the second number value

// find char x again
iPosX = strFilterValue.Find('x');

// find the start of the second number
for (int i=strFilterValue.GetLength() -1; i >= 0; i--)
{
    c = strFilterValue.GetAt(i);
    iPosSecondValue = i;

    if (!IsNumber(c)) break;
}

// if there is no first value then we do not check the second condition
if (iPosSecondValue <= iPosX + 1) strSecondExp = _T("No");
else
{
    // after
    strFilterValue.Insert(strFilterValue.GetLength(), '#');

    // before
    strFilterValue.Insert(iPosSecondValue+1, '#');
}

// set first, second values & expressions

// Tokenize
int curPos= 0;
int j = 0;
CString strTemp;

while (1)
{
    strTemp = strFilterValue.Tokenize("#", curPos);

    if (strTemp == "")
    {
        break;
    }
```

```
        strTokens[j++] = strTemp;
    };

    // if first condition is set
    if (strFirstExp.CompareNoCase("No") != 0)
    {
        iFirstValue =  _ttoi(strTokens[0]);
        strFirstExp = strTokens[1];

        // if last condition is set
        if (strSecondExp.CompareNoCase("No") != 0)
        {
            iSecondValue = _ttoi(strTokens[4]);
            strSecondExp = strTokens[3];
        }
    }
    // if last condition is set
    else if (strSecondExp.CompareNoCase("No") != 0)
    {
        iSecondValue = _ttoi(strTokens[2]);
        strSecondExp = strTokens[1];
    }

    return TRUE;
}

BOOL CMDX_ReaderDlg::IsNumber(char ch)
{
    return ((ch >= '0') && (ch <= '9'));
}

BOOL CMDX_ReaderDlg::CheckCondition(UINT uiOp1, CString strCondition, UINT uiOp2)
{
    if (strCondition == "<=")
    {
        return (uiOp1 <= uiOp2);
    }
    else if (strCondition == "<")
    {
        return (uiOp1 < uiOp2);
    }
    else if (strCondition == "=")
    {
        return (uiOp1 == uiOp2);
    }
    else if (strCondition == "!=")
    {
        return (uiOp1 != uiOp2);
    }
    else if (strCondition == ">=")
    {
        return (uiOp1 >= uiOp2);
    }
```

```
    else if (strCondition == ">")
    {
        return (uiOp1 > uiOp2);
    }

    return FALSE;
}
void CMDX_ReaderDlg::OnBnClickedButtonFrameInterval()
{
    // TODO: Add your control notification handler code here

}
```

Bibliography

Abrams, H., Watsen, K. and Zyda, M. (1998) 'Three tiered interest management for large-scale virtual environment', in *Proceedings of Virtual Reality Systems and Technology (VRST) ACM*, Taipei, Taiwan, November, pp. 125-129.

Arai, F., Tanimoto, M., Fukuda, T., Shimojima, K., Matsuura, H., and Negoro, M. (1996) 'Distributed Virtual Environment for Intravascular Tele-surgery Using Multimedia Telecommunication', in *Proceedings of the 1996 Virtual Reality Annual International Symposium (ARAIS 96)*, pp. 79.

Armitage G. (2003) 'An experimental estimation of latency sensitivity in Multiplayer Quake 3', in *Proceedings of the 11th IEEE International Conference on Networks*, Sydney, Australia, November, v.49, n.11, pp. 34-38.

Bagrodia, R., Tang, K., Goldman, S., and Kumar, D. (2006) 'An accurate, scalable communication effects server for the FCS system of systems simulation environment', in *Proceedings of the 2006 Winter Simulation Conference*, Monterey, California, USA, December.

Bangun, R. A. and Beadle, H. W. P. (1997) 'A network architecture for multiuser networked games on demand', in *Proceedings of the 1997 International Conference on Information, Communications and Signal Processing*, Singapore, September, v.3, pp. 1815-1819.

Barrus, J. W., Waters, R. C. and Anderson, D. B. (1996) 'Locales and beacons: Efficient and precise support for large multi-user virtual environments', in *Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS)*, IEEE Neural Networks Council, Santa Clara, CA, USA, March, pp. 204-213.

Bauer, D., Rooney, S. and Scotton, P. (2002) 'Network infrastructure for massively distributed games', in *Proceedings of the first workshop on Network and system support for games*, Bruanschweig, Germany, April, pp. 36-43.

Baughman, N. E. and Levine, B. N. (2001) 'Cheat-proof playout for centralized and distributed online games', in *Proceedings of IEEE Infocom*, Anchorage Alaska, USA, April, v.1, pp. 22-26.

Beigbeder, T., Coughlan., R., Lusher, C., Plunkett, J., Agu, E., and Claypool, M. (2004) 'The effects of loss and latency on user performance in Unreal Tournament 2003', in *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, Portland, Oregon, USA, pp. 144-151.

Benford, S., Greenhalgh, C., Rodden, T. and Pycock, J. (2001) 'To what extent is cyberspace really a space? Collaborative Virtual Environments', *Communications of the ACM*, July, v.44, n.7, pp. 79-85.

Bernier, Y. W. (2001) 'Latency compensating methods in Client/Server in-game protocol design and optimization', in *Proceedings of the Game Developer Conference*, San Jose, CA, USA, Mar, pp. 20-24.

Bettner, P. and Terrano, M. (2001) '1500 Archers on a 28.8: Network programming in Age of Empires and beyond', *Gamasutra Magazine*, March, Available at http://www.gamasutra.com/features/20010322/terrano_01.htm (accessed: 10 December 2001)

Bhola, S., Banavar, G. and Ahamad, M. (1998) 'Responsiveness and consistency tradeoffs in interactive groupware', in *Proceedings of the 7th ACM Conference on Computer Supported Cooperative Work*, Seattle, Washington, USA, November, pp. 79-88.

Bickmore, T. W., Cook, L. K., Churchill, E. F. and Sullivan, J. W. (1998), 'Animated autonomous personal representatives', in *Proceedings of the Second International Conference on Autonomous Agents*, Minneapolis, Minnesota, USA, May 10-13, pp. 8-15.

Blizzard (2005) 'Blizzard Entertainment', Available at http://www.blizzard.com (accessed: 12 December 2005)

Blow, J. (1998) 'A look at latency in networked games', *Game Developer Magazine*, July.

Blumberg, B. M. (1996) *Old Tricks, New Dogs: Ethology and Interactive Creatures*, MIT Media Laboratory PhD Thesis, September.

Blumberg, B., Downie, M., Ivanov, Y., Berlin, M., Johnson, M. P. and Tomlinson, B. (2002) 'Integrated learning for interactive synthetic characters', in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, San Antonio, Texas, pp. 417-426.

Bormann, C., Burmeister, C., Degermark, M., Fukushima, H., Hannu, H., Jonsson, E., Hakenberg, R., Koren, T., Le, K., Liu, Z., Martensson, A., Miyazaki, A., Svanbro, K., Wiebke, T., Yoshimura, T., and Zheng, H. (2001) 'Robust Header Compression (ROHC): framework and four profiles: RTP, UDP ESP and Uncompressed', *RFC 3095*, July.

Broadcasting (2006) 'Broadcasting (computing)', Available at http://en.wikipedia.org/wiki/Broadcasting_%28computing%29 (accessed: 15 October 2006)

Burnham, V. (2001) *Supercade: a Visual History of The Videogame Age, 1971-1984*, MIT Press, Cambridge, Massachusetts, London, England.

Cai, W., Lee, F. B. S., and Chen, L. (1999) 'An Auto-adaptive Dead Reckoning algorithm for distributed interactive simulation', in *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation*, Atlanta, Georgia, USA, pp. 82-89.

Calvin, J., Seeger, J., Troxel, G. and Hook, D. V. (1995) 'STOW real-time information transfer and networking system architecture', in *Proceedings of the 12th DIS Workshop*, March.

Carson, M. and Santay, D. (2003) 'NIST Net: a Linux-based network emulation tool', *ACM SIGCOMM Computer Communication Review*, v.3, n.3, pp. 111-126.

Cassell, J., Pelachaud, C., Badler, N., Steedman, M., Achorn, B., Becket, T., Douville, B., Prevost, S. and Stone, M. (1994) 'Animated conversation: rule-based generation of facial expression, gesture & spoken intonation for multiple conversational agents', in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, July, pp. 413-420.

Chatterjee, S. and Yang, S. C. (2005) 'Managing the optical networking solution: Is increased bandwidth enough to reduce delays?', *IT Professional*, January, v.7, n.1, pp. 46-50.

Chattopadhyay, S., Bhandarkar, S. M., and Li, K. (2005) 'Efficient compression and delivery of stored motion data for avatar animation in resource constrained devices', in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, Monterey, CA, USA, pp. 235-243.

Cheshire, S. (1996a) 'It's the latency, stupid', Available at http://www.stuartcheshire.org (accessed: 11 November 2001)

Cheshire, S. (1996b) 'Latency and the quest for interactivity', *A White Paper Commissioned by Volpe Welty Asset Management, L.L.C.*, for the Synchronous Person-to-Person Interactive Computing Environments Meeting, Available at http://www.stuartcheshire.org (accessed: 12 January 2002)

Chu, Y. H., Rao, S. G., Seshan, S. and Zhang, H. (2000) 'A case for end system multicast' , in *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Santa Clara, USA, June, pp. 1-12.

Cohen, D. (1994) 'DIS – Back to basics', in *Proceedings of the 11th Workshop on Standards for Distributed Interactive Simulations*, September, pp. 603 – 617.

Cronin, E., Filstrup, B. and Kurc, A. R. (2001) 'A distributed multiplayer game server system', *UM EECS589 Course Project Report*, May, Available at http://www.eecs.umich.edu/~bfilstru/quakefinal.pdf (accessed: 22 October 2002)

Cronin, E., Filstrup, B., Kurc, A. R. and Jamin, S. (2004) 'An efficient synchronization mechanism for mirrored game architectures', in *Proceedings of the First Workshop on Network and System Support for Games*, May, v.23, n.1, pp. 7-30.

Cheeseman, C. P. (1992) *Moving Platform Simulator III: An Enhanced High-Performance Real-Time Simulator with Multiple Resolution Display and Lighting*, Master's Thesis, Naval Postgraduate School, Monterey, California, March.

Damitio, M., Turner, S. J., Booth, C. J. M., Kirton, M. J., Milner, K. R., and Hoare, P. R. (1994) 'Comparing the breathing time buckets algorithm and the time warp operating system on a transputer architecture', in *Proceedings of SCS European Simulation Multiconference*, pp. 141-145.

Damer, B., Dipaola, S., Paniaras, J., Parsons, K., Roel, B. and Ma M. (1997) 'Putting a human face on cyberspace (panel): designing avatars and the virtual worlds they live in', in *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, Los Angeles, CA, USA, August 3-8, pp. 462-464.

Das, T. K., Singh, G., Mitchell, A., Kumar, P. S. and McGee, K. (1997) 'NetEffect: a network architecture for large-scale multi-user virtual worlds', in *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, Lausanne, Switzerland, September, pp. 157-163.

Deering, S. (1989) 'Host extensions for IP multicasting', *RFC 1112*, Information Science Institute, Marina Del Rey, CA, August.

Dijkstra, E. W. (1959) 'A note on two problems in connexion with graphs', *Numerische Mathematik,* v.1, n.1959, pp. 269–271.

Diot, C. and Gautier, L. (1999) 'A distributed architecture for multiplayer interactive applications on the Internet', *Network*, IEEE, July-Aug, v.13, n.4, pp. 6-15.

DIS (1993) 'IEEE standard for information technology – protocols for distributed simulation applications: Entity information and interaction', *IEEE Standard 1278 – 1993*, New York: IEEE Computer Society.

DIVE (2003) 'The DIVE Home Page', Available at http://www.sics.se/dive/ (accessed: 9 June 2004)

Dodson, S. (2004) 'Let the games begin', *The Guardian*, 16 November, Available at http://education.guardian.co.uk/evaluate/story/0,14726,1351738,00.html (accessed: 12 February 2006)

Doom World (2003) 'Doomworld -- The definitive source for Doom news, information and development', Available at http://doomworld.com/ (accessed: 15 December 2004)

Fichten, M. A. and Jennings, D. H. (1988) *Meaningful Real-Time Graphics Workstation Performance Measurements,* Master's Thesis, Naval Postgraduate School, Monterey, California, December.

Forouzan, B. A. (2006) *TCP/IP Protocol Suite 3rd edition*, McGrawHill Higher Education, New York, NY, USA.

Francis, P., Jamin, S., Jin, C., Jin, Y., Paxson, V., Raz, D., Shavitt, Y. and Zhang, L. (2001) 'IDMaps: A global internet host distance estimation service', *IEEE/ACM Transactions on Networking*, October , v.9, n.5, pp. 525.540.

Fritsch, T., Ritter, H., and Schiller, J. (2005) 'The effect of latency and network limitations on MMORPGs (A Field Study of Everquest2)', in *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, Hawthorne, NY, USA, pp. 1-9.

Funkhouser, T. A. (1995) RING: 'A client-server system for multiuser virtual environments', in *Proceedings of 1995 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, Monterey, California, USA, April, pp. 85-92.

Gautier, L. and Diot. C. (1998) 'Design and evaluation of MiMaze a multi-player game on the Internet', in *Proceedings of Multimedia Computing and Systems IEEE International Conference*, Austin Texas, USA, 28 June-1 July, pp. 233-236.

Gautier, L., Diot, C. and Kurose, J. (1999) 'End-to-end transmission control mechanisms for multiparty interactive applications on the Internet', in *Proceedings of IEEE Infocom 1999*, New York, NY, USA, March, v.3, pp. 1470-1479.

Gossweiler, R., Laferriere, R. J., Keller, M. L. and Pausch, R. (1994) 'An introduction tutorial for developing multi-user virtual environments', *PRESENCE: Teleoperators and Virtual Environments*, Fall, v.3, n.4, pp. 255-264.

Goyeneche, J. (1999) 'Multicast: from theory to practice', *Linux Journal*, v.1999, n.65, September.

Greenbug, S. and Marwood, D. (1994) 'Real time groupware as a distributed system: Concurrency control and its effect on the interface', in *Proceedings of ACM Conference on Computer Supported Cooperative Work*, Chapel Hill, North Carolina, USA, pp. 207-217.

Greenhalgh, C. and Benford, S. (1995) 'MASSIVE: A collaborative virtual environment for teleconferencing', *ACM Transactions on Computer-Human Interaction*, September, v.2, n.3, pp. 239-261.

Greenhalgh, C. and Benford, S. (1997) 'Boundaries, awareness, and interaction in collaborative virtual environments', in *Proceedings of the Sixth IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*, , IEEE Computer Society, Cambridge, MA. June, pp. 193-198.

Grunwald, D., Neufeld, M., Jain, and A. Gruteser, M. (2006) 'Communet' Available at http://systems.cs.colorado.edu/Networking/CommuNet/ (accessed 15 October 2006)

Haddad, I. and Gordon, D. (2002) 'Network Simulator 2: a simulation tool for Linux', *Linux Journal*, October, Available at http://www.linuxjournal.com/article/5929 (accessed: 21 December 2003)

Hagsand, O. (1996) 'Interactive multiuser VEs in the DIVE system', *IEEE Multimedia* January , v.3, n.1, pp. 30-39.

Harvey, W. (1997) 'The future of internet games, modeling and simulation: Linking entertainment and defense', *Computer Science and Telecommunications Board*, National Research Council. Washington, DC: National Academy Press, pp. 140-143.

Helder, D. A. and Jamin, S. (2002) 'End-host multicast communication using switch-tree protocols', in *Proceedings of GP2PC*, May, pp. 419.

Heng, S. Y. (2005) 'Online gaming subscription revenue surpassed US$1 Billion in 2004 and Will More Than Double by 2009', *IDC*.

Herz; J. C. (1997) 'Joystick Nation: How Videogames Ate Our Quarters, Won Our Hearts, and Rewired Our Minds', *Little Brown & Co (T)*, Princeton, NJ, USA, June.

id Software (2003) 'Doom', Available at http://www.idsoftware.com/ (accessed: 3 March 2004)

Jefferson, D. R., and Sowizral, H. (1982) 'Fast concurrent simulation using the time warp mechanism, Part I: Local Control', *Rand Note N- 1906AF*, Rand Corp., Santa Monica, Cal., December.

Jefferson, D. R. (1985) 'Virtual time', *ACM Transactions on Programming Languages and Systems*., July, v.7, n.3, pp. 404–425.

Jehaes, T., Vleeschauwer, D. D, Doorselaer, B. V., Deckers, E., Coppens, T., Naudts, W., Spruyt, K., and Smets, R. (2003) 'Access network delay in networked games', in *Proceedings of the 2nd Workshop on Network and System Support for Games*, Redwood City, California, USA, pp. 63-71.

Jiang, X., Safaei, F. and Boustead, P. (2005) 'Latency and scalability: a survey of issues and techniques for supporting networked games', in *Proceedings of the 13th IEEE International Conference on Networks jointly held with the 7th IEEE Malaysia International Conference on Communications*, Kuala Lumpur, Malaysia, pp. 150-155.

Kanamaru, A., Ohta, K., Kato, N., and Mansfield, G. (2000) 'A simple packet aggregation technique for fault detection', *International Journal of Network Management*, July-August, v.10, n.4, pp. 215-228.

Keshav, S. (1997) 'Simulator for an engineering approach to computer networking', Available at http://www.cs.cornell.edu/skeshav/real/ (accessed: 22 October 2003)

Kim, J. (2000) *Protocol*, Data Communication Research Centre, Seoul, Korea.

Kornaros, G., Papaefstathiou, I., Nikologiannis, A. and Zervos, N. (2003) 'A fully-programmable memory management system optimizing queue handling at multi gigabit rates', in *Proceedings of the 40th Conference on Design Automation*, Anaheim, CA, USA, pp. 54-59.

Lamport, L. (1978) 'Time, clocks, and the ordering of events in a distributed system', *Communications of the ACM*, July, v.21, n.7, pp. 558-565.

Lea, R., Honda, Y., and Matsuda, K. (1997) 'Virtual society: Collaboration in 3D spaces on the internet', *Computer Supported Cooperative Work*, v.6, n.2-3, pp. 227-250.

Leigh, J., Johnson, A. E. and DeFanti, T. A. (1997) 'CAVERN: A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments', *Journal of Virtual Reality Research, Development and Applications, the Virtual Reality Society*, v.2, n.2, pp. 217-237.

Lin, K. (1995) 'Dead reckoning and distributed interactive simulation', in *Proceedings of SPIE Conference (AeroSense'95)*, Orlando Florida, April.

Lincroft, P. (1999) 'The internet sucks: Or, what I learned coding X-Wing vs. TIE Fighter', in *Proceedings of Game Developers Conference*, March.

Liu, C. (2000) 'Multimedia over IP: RSVP, RTP, RTCP, RTSP', Available at http://www2.ing.puc.cl/~jnavon/IIC3582/Present/3/mmip.htm (accessed: 12 December 2002)

Loral Systems Company (1992) 'Strawman distributed interactive simulation architecture description document volume 1', *Advanced Distributed Simulation Technology Program Office*, Orlando, Florida, March.

Loyall, A. B. and Bates, J. (1993) 'Real-Time control of animated broad agents', in *Proceedings of Fifteenth Annual Conference of the Cognitive Science Society*, Boulder, Colorado, June.

Lubachevsky, B., Shwartz, A., and Weiss, A. (1989) 'Rollback sometimes works...if filtered', in *Proceedings of the 21st Conference on Winter Simulation*, Washington, D.C., USA, pp. 630-639.

Macedonia, M. R., Zyda, M. J., Pratt. D. R., Lewis, T., Boger, D. and Soversign, M. G. (1994) 'NPSNET: A network software architecture for large-scale virtual environments', *Presence*, MIT Press, Fall, v.3, n.4, pp. 265-287.

Macedonia, M. R., Brutzman, D. P., Zyda, M. J., Pratt, D. R., Barham, P. T., Falby, J. and Loche, J. (1995) 'NPSNET: A multi-player 3D virtual environment over the internet', in *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, NewYork, April, pp. 93-94.

MacKenzie, I. S., and Ware, C. (1993*)* 'Lag as a determinant of human performance in interactive systems', in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Amsterdam, The Netherlands, pp. 488-493.

Makki, A. (2006) 'Adaptive modelling of IP packet aggregation', in *Proceedings of London Communication Symposium 2006*, London, UK, September.

Mastaglio, T. W. and Callahan, R. (1995) 'A large-scale complex virtual environment for team training', *IEEE Computer*, July, v.28, n.7, pp. 49-56.

Mauve, M., Hilt, V., Kuhmunch, C., and Effelsberg, W. (1999) 'An application-layer protocol for the transmission of interactive media with real-time characteristics', in *Proceedings of IEEE Multimedia Systems '99 (ICMCS'99)*, Florence, Italy, June.

Mauve, M. (2000) 'How to keep a dead man from shooting'*,* in *Proceedings of the 7th International Workshop on Interactive Distributed Multimedia Systems*, pp. 199-204.

Mauve, M., Vogel, J., Hilt, V. and Effelsberg, W. (2004) 'Local-lag and Timewarp: Providing consistency for replicated continuous applications', *IEEE Transaction on Multimedia*, v.6, pp. 47-57.

McDonald, C. (1991) 'The cnet network simulator', Available at http://www.csse.uwa.edu.au/cnet/ (accessed: 12 November 2003)

McLean, H. (2004) 'Welcome to play school', *The Guardian*, 16 November, Available at http://education.guardian.co.uk/elearning/story/0,,1351965,00.html (accessed: 12 February 2006)

Medhi, D. (1999) *Network Reliability and Fault Tolerance*, Wiley Encyclopedia of Electrical and Electronics Engineering, John Wiley, New York.

Miller, D. C. and Thorpe, J. A. (1995) 'SIMNET: The advent of simulator networking', in *Proceedings of the IEEE*, August, v.83, n.8, pp. 1114-1123.

Mogul, J. (1984) 'Broadcasting internet datagrams'*, RFC 919*, Information Sciences Institute, Marina Del Rey, CA, October.

Moon, K. S., Muthukkumarasamy, V., Nguyen A. T., and Kim, H. S. (2005) 'Maintaining consistency in distributed network games', in *Proceedings of the 13th IEEE International Conference on Networks jointly held with the 7th IEEE Malaysia International Conference on Communications*, Kuala Lumpur, Malaysia, pp. 374-379.

Moon, K. S., Muthukkumarasamy, V. and Nguyen A. T. (2006a) 'Efficiently maintaining consistency using tree-based P2P network system in distributed network games', in *Proceedings of The Edutainment 2006, International Conference on E-learning and Games,* Hangzhou, China, LNCS 3942, April, pp. 648-657.

Moon, K. S., Muthukkumarasamy, V. and Nguyen A. T. (2006b) 'Reducing network latency on consistency maintenance algorithms in distributed network games', in *Proceedings of IADIS International Conference, Applied Computing 2006,* San Sebastian, Spain, v.1, pp. 6.

Moon, K. S., Muthukkumarasamy, V. and Nguyen A. T. (2006c) 'Reducing bandwidth requirements of consistency maintenance algorithms in distributed network games', in *Proceedings of the International Conference on Computer and Communication Engineering, ICCCE'06,* Kuala Lumpur, Malaysia, May, v.l, pp 396-401.

Morse, K. L. (1996) 'Interest management in large-scale distributed simulations', *Technical Report ICS-TR-96-27*, University of California, Irvine.

Munson, J. and Dewan, P. (1996) 'A concurrency control framework for collaborative systems', *In ACM CSCW*, pages 278-287.

Ng, Y. (1997) 'Designing fast-action games for the internet', Available at http://www.gamasutra.com/features/19970905/ng_01.htm (accessed: 23 October 2004)

Nichols, J. and Claypool, M. (2004) 'The effects of latency on online madden NFL football', in *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, Kinsale, County Cork, Ireland, June 16-18, pp. 146-151.

Oliver, M. R. and Stahl, D. J. (1987) *Interactive, Networked, Moving Platform Simulators,* Master's Thesis, Naval Postgraduate School, Monterey, California, December.

Pantel, L and Wolf, L. C. (2002) 'On the impact of delay on real-time multiplayer games', in *Proceedings of the 12th International Workshop on Network and Operating Systems Supported for Digital Audio and Video*, Miami, Florida, USA, pp. 23-29.

Papoulis, A. (1984) '*Poisson Process and Shot Noise*', McGraw-Hill, New York, USA.

Peden, M. and Young, G. (2001) 'From voice-band modems to DSL technologies', *International Journal of Network Management*, September, v.11, n.5, pp. 265-276.

Pellegrino, J. D. and Dovrolis, C. (2003) 'Bandwidth requirement and state consistency in three multiplayer game architectures', in *Proceedings of the 2nd Workshop on Network and System Support for Games*, Redwood City, California, USA, pp. 52-59.

Perkins, C. and Crowcroft, J. (2000) 'Notes on the use of RTP for shared workspace applications', *ACM Computer Communication Review*, v.30, n.2, pp. 35-40.

Poole, S. (2000) 'Trigger Happy: Videogames and the entertainment revolution' *London: 4th Estate*.

Pope, A. (1989*)* 'The SIMNET network and protocols', *Technical Report 7102*, Cambridge, MA: BBN Systems and Technologies, July.

Postel, J. (1980) 'User Datagram Protocol', *RFC 768*, August.

Prakash, A. and Shim, H. S. (1994) 'DistView: Support for building efficient collaborative applications using replicated objects', in *Proceedings of ACM Conference on Computer Supported Cooperative Work*, Chapel Hill, North Carolina, USA, pp. 153-164.

Pratt, D. R. (1993) *A Software Architecture for the Construction and Management of Real-Time Virtual World,* PhD Thesis, Naval Postgraduate School, Monterey, CA, June.

Pullen, J. M. and Wood, D.C. (1995) 'Networking technology and DIS', *in Proceedings of the IEEE*, August, v.83, n.8, pp. 1156-1167.

Quax, P., Monsieurs, P., Vleeschauwer, D. D., Lamotte, W., and Degrande, N. (2004) 'Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game', in *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support for Games*, Portland, Oregon, USA, pp. 152-156.

Ratnasamy, S., Ermolinskiy, A., and Shenker, S. (2006) 'Revisiting IP multicast', in *Proceedings of SIGCOMM' 06*, Pisa, Italy, September, pp. 15-26.

Razafindralambo, T., Lassous, I. G., Iannone, L., and Fdida, S. (2006) 'Dynamic packet aggregation to solve performance anomaly in 802.11 wireless networks', in *Proceedings of the 9th ACM International Symposium on Modeling Analysis and Simulation of Wireless and Mobile Systems*, Terromolinos, Spain, pp. 247-254.

Russo, K., Schuette, L., Smith, J., and McGuire, M. (1995) 'Effectiveness of various new bandwidth reduction techniques in ModSAF', in *Proceedings of the 13th DIS Workshop on Standards for the Interoperability of Distributed Simulation*, *Institute for Simulation and Training*, Orlando, September, pp. 587-591.

Salen, K., and Zimmerman, E. (2003) 'Rules of play: game design fundamentals', *MIT Press*, Cambridge, Massachusetts, London, England.

Sarin, S. and Greif, I. (1985) 'Computer-based real-time conferencing systems', *IEEE Computer*, October, v.18, n.10, pp. 33-45.

Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. (1996) 'RTP: A transport protocol for real-time applications', *RFC 1889*, January 25.

SGI (2005) 'SGI TPL View (6 dog)', Available at http://techpubs.sgi.com/library/tpl/ (accessed: 15 October 2005)

Sharkey, P. M., Ryan, M. D. and Roberts, D. J. (1998) 'A local perception filter for distributed virtual environments', *IEEE Virtual Reality Annual International Symposium (VRAIS 98)*, Atlanta, GA, March 14-16, pp. 242-249.

Sheldon, N., Girard, E., Borg, S., Claypool, M., and Agu, E. (2003) 'The effect of latency on user performance in Warcraft III', in *Proceedings of the 2nd Workshop on Network and System Support for Games*, Redwood City, California, USA, pp. 3-14.

Shuster, L. (2003) 'Global gaming industry now a whopping $35 billion market', *Compiler*, July.

Singh, G., Serra, L., Png, W. and Ng, H. (1999) 'BrickNet: A software toolkit for network-based virtual worlds', *Presence: Teleoperators and Virtual Environments*, Winter, v.3, n.1, pp. 19-34.

Singhal, S. K. (1996) *Effective Remote Modelling in Large-Scale Distributed Simulation and Visualization Environments*. PhD Thesis, Department of Computer Science, Stanford University, Palo Alto, August.

Singhal, S. K. and Cheriton, D. R. (1995) 'Exploiting position history for efficient remote rendering in networked virtual reality', *Presence: Teleoperations and Virtual Environments*, Spring, v.4, n.2, pp. 169-193.

Singhal, S. K. and Zyda, M. (1999) *Networked Virtual Environments: Design and Implementation*, Addison Wesley, ACM Press, New York, USA, July.

Smed, J., Kaukoranta, T. and Hakonen, H. (2001) 'Aspects of networking in multiplayer computer games', in *Proceedings of International Conference on Application and Development of Computer Games in the 21st Century*, Hongkong, September.

Smed, J., Kaukoranta, T. and Hakonen, H. (2002) 'A review on networking and multiplayer computer games', *Technical Report 454*, Turku Centre for Computer Science, Turku, Finland, April.

Smith, D. B. and Streyle, D. (1987*) An Inexpensive Real-Time Interactive Three-Dimensional Flight Simulator System*, Master's Thesis, Naval Postgraduate School, Monterey, California, June.

Stabell, B. and Schouten, K. R. (2001) 'The story of XPilot', *ACM Crossroads*, January.

Stallings, W. (1997) *Data and Computer Communications*, 5th ed., Prentice-Hall, Upper Saddle River, NJ, USA.

Steinman, J. S. (1993) 'Breathing time warp', in *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation*, San Diego, California, May , pp. 109-118.

Steinman, J. S., Wallace, J. W., Davani, D. and Elizandro, D. (1998) 'Scalable distributed military simulations using the speeds object-oriented simulation framework', in *Proceedings of Object-Oriented Simulation Conference (OOS'98)*, pp 3-23.

Steinman, J. S. and Weiland, F. (1994) 'Parallel proximity detection and the distribution list algorithm', in *Proceedings of the 1994 Workshop on Parallel and Distributed Simulation*, Edinburgh, Scotland, United Kingdom, pp. 3-11.

STOW (1998) 'STOW 98 event final report', Available at http://stow98.spawar.navy.mil/STOW98EventExSum/index.html (accessed: 15 Semtember 2001)

Sung, U., Yang, J., and Wohn, K. (1999) 'Concurrency control in CIAO', in *Proceedings of the IEEE Virtual Reality*, March, pp. 22-28.

Tamagawa, K., Yamada, T., Ogi, T. and Hirose, M. (2001) 'Developing a 2.5-D video avatar', *Signal Processing Magazine*, IEEE, May, v.18, n.3, pp. 35 –42.

The QuakeForge Project (2003) 'QuakeForge', Available at http://www.quakeforge.net/ (accessed: 9 April 2004)

The Straight Dope (2002) 'Is "Dead reckoning" short for "Deduced reckoning"?', *The Straight Dope, 21, November*, Available at http://www.straightdope.com/mailbag/mdeadreckoning.html (accessed: 27 October 2006)

Taylor, D. E., Herkersdorf, A., Doring, A., and Dittmann, G. (2005) 'Robust header compression (ROHC) in next-generation network processors', *IEEE/ACM Transactions on Networking (TON)*, August, v.13, n.4, pp. 755-768.

Tye, C. S. and Fairhurst, G. (2003) 'A review of IP packet compression techniques', Available at www.cms.livjm.ac.uk/pgnet2003/submissions/Paper-13.Pdf (accessed: 11 July 2004)

Ultima Online (2003) 'ORIGIN - Ultima Online', Available at http://www.uo.com (accessed: 28 May 2004)

Vogel, J. and Mauve, M. (2001) 'Consistency control for distributed interactive media', in *Proceedings of the Ninth ACM International Conference on Multimedia*, Ottawa, Canada, October, pp. 221-230.

Watagodakumbura, C., Jennings, A., and Shenoy, N. (2003) 'Effects of traffic aggregation on quality of service parameters', in *Proceedings of the 2003 IFIP/ACM Latin America Conference on Towards a Latin American Agenda for Network Research*, La Paz, Bolivia, pp. 163–172.

Waters, R. C., Anderson, D. B. and Schwenke, D. L. (1997) 'Design of the interactive sharing transfer protocol', *WET ICE '97 -- IEEE Sixth Workshops on Enabling*

Technologies for Collaborative Enterprises: Infrastructure for Collaborative Enterprises, IEEE Computer Society Press, Los Alamitos, CA, June 1997, pp. 140-147.

Weeks Jr., G. K. and Phillips Jr., C. E. (1989) *The Command and Control Workstation of the Future: Subsurface and Periscope Views*, Master's Thesis, Naval Postgraduate School, Monterey, California, June.

Winn, M. C. and Strong, R. P. (1989) *Moving Platform Simulator II: A Networked Real-Time Simulator with Intervisibility Displays,* Master's Thesis, Naval Postgraduate School, Monterey, California, June.

Wikipedia (2006) 'Wikipedia, the free encyclopedia', Available at http://en.wikipedia.org/wiki/Main_Page (26 February 2006)

Wloka, M. (1994) 'Lag in multiprocessor VR'*, Presence*, Fall, v.3, n.4.

Wolf, M. J. P. (2001) 'The Medium of the Video Game', *University of Texas Press*, Texas, USA.

Wolfson, O. (1987) 'The overhead of locking (and commit) protocols in distributed databases', *ACM Transactions on Database Systems (TODS)*, September, v.12 n.3, pp.453-471.

Wray, M. and Belrose, V. (1999) 'Avatars in livingSpace', in *Proceedings of the Fourth Symposium on Virtual Reality Modeling Language*, Paderborn, Germany, February 23-26, pp. 3-19.

Yang, Y., Wang, X. and Chen, J.X. (2002) 'Rendering avatars in virtual reality: integrating a 3D model with 2D images', *Computing in Science & Engineering*, Jan.-Feb, v.4, n.1, pp. 86-91.

Yasui, T., Ishibashi, Y., and Ikedo, T. (2005) 'Influences of network latency and packet loss on consistency in networked racing games', in *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, Hawthorne, NY, USA, pp. 1-8.

Zou, L., Ammar, M. H., and Diot, C. (2001) 'An evaluation of grouping techniques for state dissemination in networked multi-user games', in *Proceedings of the Ninth International Symposium in Modeling*, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'01), pp. 33-40.