BOND UNIVERSITY
SCHOOL OF INFORMATION TECHNOLOGY

# Efficient Methods for Improving Scalability and Playability of Massively Multiplayer Online Game (MMOG)

Kusno Prasetya

BIT (Sekolah Tinggi Teknik Surabaya, Indonesia), MIT (Hons) (Bond)

A dissertation submitted in fulfilment of the requirements of the degree of

Doctor of Philosophy for the School of Information Technology, Bond University.

February 2010

Typeset in $\LaTeX\,2_\varepsilon$

## Statement of Originality

Submitted for examination: 2010

Approved for the degree of Doctor of Philosophy: 2010

# Abstract

The popularity of Massively Multiplayer Online Games (MMOG) is increasing rapidly these days with more players prefering to play with or against human players. Having more than one player in the game at one time enables people to use MMOGs to socialize with others while at the same time getting the enjoyment of playing games. However, game service providers often face the challenges caused by network latency and player's behaviours. Despite of having fast development in internet technology, network latency is still one of the main problems in networking part of MMOGs. Network latency disrupts gameplay experience by causing game state inconsistencies amongst game server and players machine which in the end, it discourages players to play the MMOGs. Meanwhile, cheating in MMOGs has been a constant problem which also causes game state inconsistencies with the game mechanics. There are various ways to cheat in MMOG and one of them is using bot to automatically control player to do certain tasks. In the end, cheating using bot affects playability of an MMOGs by disrupting game balance and eliminating one of the main purposes of playing a MMOG: to play against other player. Research in MMOG is a relatively new field, with few proposals to improve scalability and address cheating problems. There are ways to improve scalability such as by improving the internet itself or adding more game servers and increasing bandwidth. However, this kind of solution often incurs additional operational costs for game service provider and takes time to be implemented in internet standard. Also, this solution does not solve the problem in player's side where the same problem by network latency could happen. Meanwhile, researches to prevent cheating in MMOG have resulted in few proposals about how to detect cheaters and prevent them to play the game. There are also other methods devised by game service provider to deter cheaters before they even login into the game. However, the success of their methods is somewhat limited and lack of flexibility. Therefore, there are potentials for improvement for existing research or to solve the problem through another perspective. This thesis proposes a combination of solutions to improve scalability and playability of MMOGs. To improve scalability, this research presents Game World Partition (GWP) which categorized as a part of Interest Management System (IMS). GWP is a method commonly implemented in MMOG where it divides game world into partitions and manage the communication between players. Player in one partition does not need to communicate with other players in different partitions. This method improves scalability by reducing the number of packets transferred during gameplay and thus, allows more players to play in one game session. The research work proposes a new GWP method which is simple to implement but still offers improvement in scalability. This research also proposes a network workload evaluation method that could assist game service provider or programmer in evaluating and predicting their network resource requirements. Furthermore, this research proposes the use of Artificial Neural Network (ANN) for Dead Reckoning (DR) in MMOG. DR is a mathematical model that can be used to extrapolate player's location based on the previous locations. DR improves playability by providing smoother gameplay whenever network latency occurs. To address cheating problems in MMOG, this research proposes an extension of ANN for DR to detect player movement generated by bot. The bot detection system analyse one player's movement and determine the possibility of a player being controlled by bot or human. Results from experiments are presented in each of the solutions described in this thesis. The experiment uses random-generated data and data from real games whenever possible. Comparisons with commonly implemented method in MMOG shows how the solutions proposed in this thesis perform through simulations.

# Acknowledgements

This PhD thesis would not have been possible without the support of many people and organisations to whom I owe a great deal. I would like to thank particularly:

*Dr. Zheng da Wu*, for his belief in my abilities, his advice, his encouragement, his wisdom, his patience and his endless kindness. He always supported me in many ways and our weekly discussions trigger a lot of new ideas for my research.

*The School of Information Technology, Bond University*, for providing me with resources and opportunity to complete my PhD, as well as introducing me to and allowing me to pursue my teaching career. Special thanks to the head of School of IT, *Dr. Iain Morisson* and *Kim Younger* to make everything I have done so far possible.

*The Australian Government*, for providing me with an International Postgraduate Research Scholarship (IPRS) Award during my PhD candidature.

*Dr. Marcus Randall and Sam Gauthier*, for giving me the chance to become the tutor of Information Technology 1 course. Especially for Sam, her willingness to listen and give me advice on tutoring helped me to find the passion and keep learning about teaching.

To fellow PhD candidates *Pedro Gómez, Emma Chávez and Percy Pari Salas*, whose advices, suggestions and friendship keep giving me motivation to move forward. So much of this thesis would not have been possible without them.

To many of my *students* who I have taught since I began teaching 3 years ago. You all gave me positive experiences and inspirations over the time I spent during the PhD.

To my *family in Indonesia*, for providing me great deal of love and motivation which help me to keep on going through all these years. They are the best family one could ever get and I am grateful to have them as my family.

At least but not last, to my *wife, Min Jee Kim* and my *daughter Christina Grace Prasetya*, who always encourage me througout the journey of pursuing my doctoral studies. I am very grateful to my wife's unselfish blessings and support. This thesis is defnitely dedicated to both of them.

## Publications Arising from this Research

1. K. Prasetya and Z. D. Wu. Performance analysis of game world partitioning methods for multiplayer mobile gaming. In *NetGames 2008: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 72-77, New York, NY, USA, 2008. ACM.

2. K. Prasetya and Z. D. Wu. A Mechanism for Network Workload Evaluation in MMOGs. In *International Journal of Computer Games Technology*, New York, NY, USA. Hindawi Publishing Corporation. (Under review)

3. K. Prasetya and Z. D. Wu. Analysis of Recurrent Neural Network for Dead Reckoning in 3D Multiplayer Online Games. In *International Journal of Computer Games Technology*, New York, NY, USA. Hindawi Publishing Corporation. (Under review)

4. K. Prasetya and Z. D. Wu. Artificial Neural Network for Bot Detection System in MMOGs. To be submitted to NetGames 2010.

# Contents

# List of Figures

# Chapter 1

# Introduction

The gaming technology is advancing rapidly these days, now consumers (game players) have better and wider option of gaming. At the same time, broadband Internet is becoming more available to everyone and the concept of constantly connected to the network also becomes a reality. The result of these two technologies leads to advances in multiplayer gaming. Before, multiplayer gaming was limited to a Local Area Network (LAN) which also limits the location of players. In LAN games, players have to be together at the same place. Using broadband Internet, it is possible to have players from different and faraway places to play together in a multiplayer game. Thus, Multiplayer Online Game (MOG) was born and started gaining popularity.

The broadband Internet is also becoming cheaper over the years and thus, transferring or downloading a large amount of data is no longer a problem for most people with the current Internet. Therefore, the Internet nowadays can support multiplayer gaming with more complex mechanism. As a comparison, Multi-User Dungeon (MUD) which is one of the predecessors in multiplayer online game could only offer text-based games. Recent

MOG offers 3D animation and special effects (sfx), giving better video/audio feedback to the players. Furthermore, the improvement in the Internet also makes it capable to support more number of players to be connected in one game. With this, the phenomena of Massively Multiplayer Online Games (MMOG) started [41].

Massively Multiplayer Online Game (MMOG) is a relatively new type of game where a large number of players can play together with others through the Internet. MMOG offers a new and better gaming experience in the multiplayer gaming area. On the contrary, single player games usually rely on computer controlled character to provide game interaction with the player and there is a tendency for the game to become monotonous. The experience of playing with other human player is the main reason why MMOG's popularity is rapidly increasing since few years ago. A very good example of successful MMOG is *World of Warcraft (WoW)* [94]. The number of active player subscriptions reached 10 million players by early 2008 [93] and increased to 11.5 million by the end of 2008 [95]. Along with the increase of the number of players, we have more challenges for both game developers and game service providers to create and deploy MMOGs. There are two basic categories to examine an MMOG, they are playability and scalability. Both of them cover a wide range of factors in the game design and implementations. Also, they are related to one another. All definitions, assumptions and mechanism regarding this topic will be explained further in chapter 2.

Despite of the fact that internet has become faster and more available to the market, it is not yet perfect with many occasions of network lag and packet loss. Physical distance between player's computer and game server also contributes to the problems where farther distance will often result in larger latency and more possible packet loss. To address this issue, it is common to implement sharded game universe. It is a method where we have multiple copy of the MMOG in different server in different locations. One server operates one game universe independently and players in that universe cannot see or contact other players in other game universe. In this way, a player can choose the best or closest server to get the lowest network latency. By solving the problems caused by network latency and packet loss, we can improve the scalability of a game so as to allow more players to play in a game server. One of the most common methods is by improving the efficiency of network resource management method in game communication protocols. With improved efficiency, bandwidth requirement can be lowered and thus could provide better playability by reducing game jitter or stutter [5].

The second key problem is cheating in MMOG. It is the part that affect playability of an MMOG. Assuming that game developer implements a fair game mechanic, all players should not have significant advantage compared to others. There are many types of cheating and they are generally involves the violation of the rules defined by game mechanic. Another type of cheating is unfair enhancement of player's skill to play the game, which is also one of our research aims in this thesis. The cheat requires a special program that we call Bot. Bot enables players to maintain accurate and autonomous player input over a long period of time.

This thesis presents our proposed solutions on how to improve scalability and playability of MMOG. They are very wide research areas, thus we limit our research objectives with problems of MMOG that are related to network latency, resource planning and cheating. The original contributions of this dissertation are as follows:

- One of the ways to minimize the impact of network latency and packet loss is to reduce the amount of required state update. Game world partition has been used in multiplayer online game to provide better network management and reduce the number of network traffic by filtering the receiver for state update. In this thesis, we define the metric for performance analysis specifically for game world partition. Furthermore, we propose our method where its design consideration is to keep game mechanics simple but also efficient in network resource usage for MMOGs. We call it Brickworks with Internal Partitions (BIP). In order to compare current methods with ours, we developed a simulator which takes random player movement pattern as the input and generate results based on the defined metrics. Our simulation shows better result for BIP compared to other traditional partition method and based on our analysis, it will not consume much of computing resources because of its simple mechanism.

- Network workload is often used to determine efficiency of network algorithms or communication protocols. In this thesis, we present our mechanism to evaluate network workload in MMOGs. The mechanism itself consists of several modules, making it easier to exchange data or extract more result for further analysis. The main purpose of our mechanism is to help game developer, service provider and other researcher to evaluate network workload. Furthermore, our mechanism can be used to perform network performance analysis and comparison and to identify key problems in different scenarios. In our implementation of game system, we use Game World Partition (GWP) as the Interest Management System and recorded data from the game *Quake II* as an example about how our mechanism can be useful for network workload evaluation. The result of our simulation has proven that we can use our mechanism to identify the impact of different parameters in our game system.

- Dead Reckoning is a well known method to minimize the impact of network delay in multiplayer gaming applications [71]. It works by extrapolating other player's location when network delay occurs. Artificial Neural Network (ANN) is a branch of Artificial Intelligence (AI) that one of its usages is in value prediction and has the ability to adapt with various input patterns. The combination of both is usable in dead reckoning due to the fact that human behaviour and game mechanics exhibit a certain pattern. By recognizing general pattern of player movement, artificial neural network can provide better prediction for player's location compared to other Dead Reckoning methods. In this thesis, we present our research result in designing and implementing artificial neural network for multiplayer games. It provides better accuracy of prediction compared to standard DR method and it offers the flexibility and possibility of expansion should the need arise in the future for more sophisticated value estimation.

- Cheating is one of the biggest and constant problems in MMOGs. Games with high frequency of cheating will surely lose its appeal to genuine and honest players who want to play the game. This is the reason why game provider these days put cheating prevention as one of the top priorities. Bot is just one way of cheating, but very efficient one. There are various methods to prevent cheating using bot. In this paper, we examine the potential of Artificial Neural Network (ANN) to detect and recognize bot from human players. We start with the assumption that one bot always acts in the similar pattern in gameplay. Meanwhile, it is much rarer to see two players with similar gameplay pattern. The result of our experiment supports our initial hypothesis with the potential for future research in order to get better results.

## 1.1 Research Objectives and Methodology

As mentioned previously, research in MMOG includes a very wide area and expertise. Hence, we need to limit our research objectives according to the four issues discussed in the previous sections. In summary, the problems covered by this dissertation can be categorized into two: problems caused by networking aspects and problems caused by cheating players. Relative to the research objectives, our proposed solutions aim to improve scalability and playability of the MMOG.

The first issue is about how to improve the efficiency of network resource usage. As we discussed before, a brute way to improve scalability is by increasing network bandwidth and adding more game servers. However, they are not efficient methods and of course, very costly.

The second issue is about how to measure and predict network workload in an MMOG. The solutions for this issue would be very useful for game developers and game service providers. Also, there is a research potential to make the solutions useful and practical for other ongoing and future research related to efficiency measurement in networking.

The third issue is about how to improve tolerance for network latency. Options for solutions are plenty in this issue. However, the main challenge is to make the solutions flexible. Current solutions work by analysing specific game mechanics and devise a mechanism to interpolate or extrapolate data. However, the solutions to one type of game mechanics may not be applicable to others. Without flexibility to cope with variety of game mechanics, developing solutions to address this issue would not be cost efficient.

The final issue is about cheat detection system in MMOG. As we discussed in the previous section, Bot usage is one of the cheat types that has no solution at the moment.

Game service providers tried to develop special software that runs in client's computer to detect whether the user is using bot or not. However, the main weakness of this solution lies in the fact that the anti-cheat software needs to run in player's machine. One general rule about security is that software is no longer safe when the software can be accessed directly by the people with malicious intent. Therefore, we are looking at the solution that runs in the game server. The challenge here is how to make the solution works while maintaining efficiency because anything runs in the server will cost game service provider even more.

To achieve our research objectives, we use simulator developed by ourselves to verify and compare our proposed solutions with other methods. Due to the differences in each issue in this thesis, we will use different methodology for each issue. In Chapter 3, we will address the network resource management issue by introducing our Interest Management System. We developed a game scenario generator where it will generate randomized players' movement patterns. Then we use the patterns to evaluate our solution. Chapter 4 will present our solution to the network workload evaluation issue. We used mathematical formulas to calculate the workload. Next, we obtained real game data and use the data in our formulas to verify its functions and results. In chapter 5, we designed and implemented an Artificial Neural Network (ANN) system for Dead Reckoning to improve the tolerance of an MMOG to network latency and packet loss. Just like any other DR methods, our method also has the potential to reduce the requirement of game state update frequency. In order to train our DR method, we modify an open-source multiplayer game so we can record player's action and movement pattern from several game sessions. To verify the result of our DR method, we use the recorded data from chapter 4 and calculate the error-rate. Finally, in chapter 6, we use the same open-source game to run two different game bots in several game sessions to train our bot detection system. Our bot detection system uses the ANN as its core to perform the bot movement and action pattern recognition. To verify the result, we use the system to recognize the input-pattern generated by the same bots in different game sessions.

## 1.2 Thesis Organization

The rest of dissertation is organized as follows. In Chapter 2, the background of the research is explained. The explanation covers game mechanics, problem definitions, literature review and research objectives. The game world partition systems are investigated in Chapter 3. Chapter 4 presents our network workload evaluation method based on the game mechanic described in Chapter 2 and using the system described in Chapter 3 as the basis of implementation example. Chapter 5 introduces the detail of our ANN method for DR in MMOG, followed by Chapter 6 that explains bot detection system using ANN. Finally, Chapter 7 concludes the dissertation with discussions about future work.

# Chapter 2

# Literature Review

Over the last decade, MMOGs have created new phenomena in gaming industry. Previously considered as an impossibility due to technology limitation, new advances in Internet and computing technologies provide the means to support MMOG with more number of players and complex game mechanics. One very good example of the increasing popularity can be seen in the number of active players in MMOGs. Figure 2.1 which is taken from [60] shows the number of active subscriptions to various MMOGs since 1997.

World of Warcraft [94], which is considered to be the most popular MMOG, has more than 11.5 millions of active subscribers since 2008 [95]. The massive number of players put massive demand on game server to provide good game service and it is usually very costly to maintain. Also, the current Internet technology that supports communication in MMOGs is not perfect. There are problems in Internet that could hinder the scalability and playability of MMOGs. There are also playability problems that come from the players who cheat in MMOGs. Currently, all of these problems are being hot topics in the research field. We survey the basic mechanism and challenges of MMOG in this chapter which leads

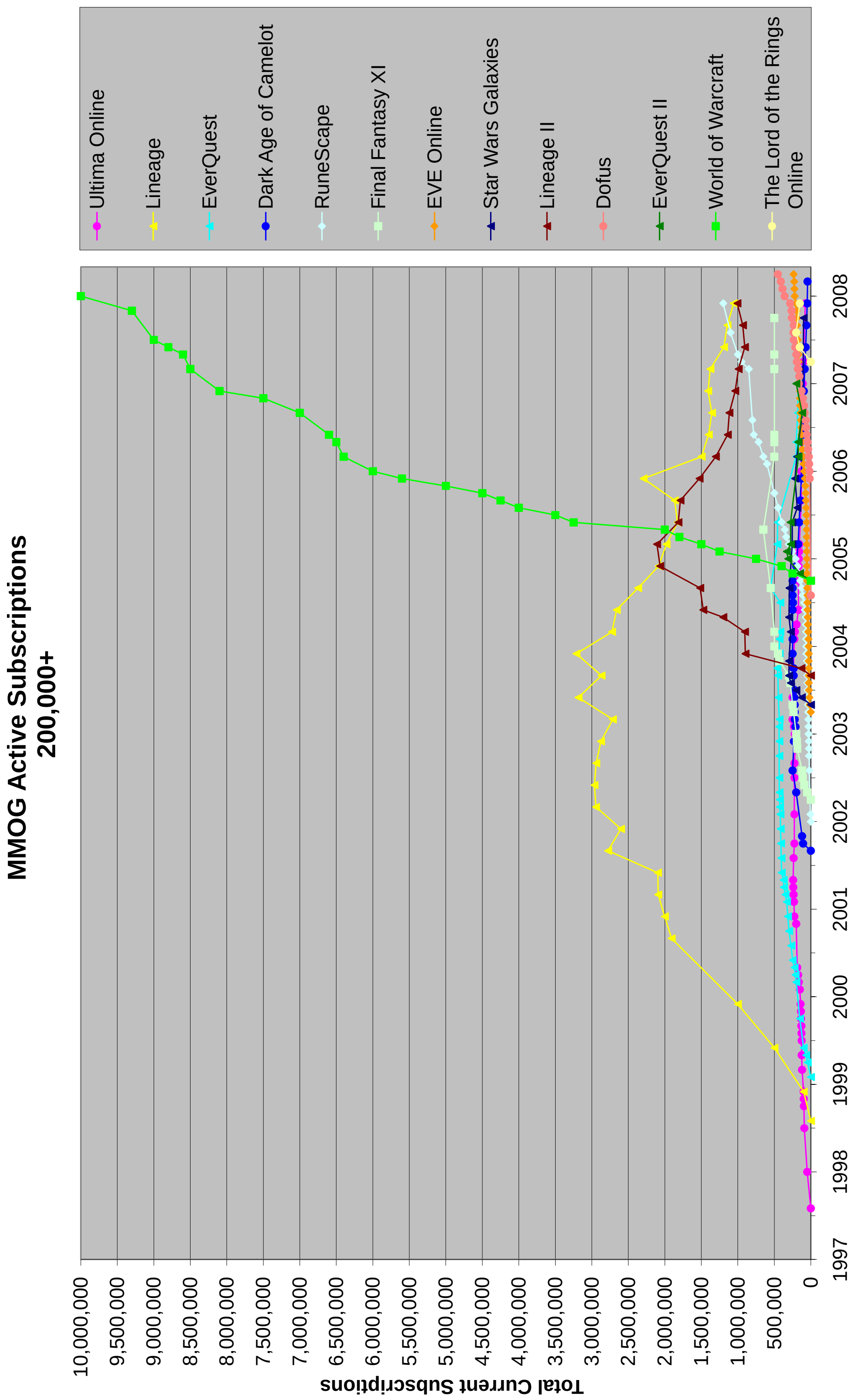

Figure 2.1: MMOG Active Subscriptions (Source: http://www.mmogchart.com)

to the research objectives of this thesis.

## 2.1 Overview of MMOG

MMOG is classified as video game. Video game is a game application that interacts with the player through a user interface on a game machine. The game machine consists of input devices to receive input from the player and output devices that give feedback to the player. Commonly, a player is represented by an avatar in the game. One of the simplest forms of video games is Single Player Games (SPG). In SPG, a player will use the platform to play a game. The game processes in SPG are the basic of all types of gaming applications. Figure 2.2 shows the general processes in an SPG.

Everything begins when the game starts where the game will receive command input from player through the game machine's input devices. Any inputs from the player will be translated into game state updates. For example, when a player presses the left button, the game will translate the input into game state update which will be processed by the game and as a result, player's avatar will move to the left. All of these processes will keep repeating itself in a loop until the game is finished either by player's intervention or by game mechanics. During gameplay, game machine which runs the game will process and store all information about the game. The game information is called *game state*. In general, one game state update might contains the following:

- Player's login/logout information.
- Player's command/action.
- Player's statistic.

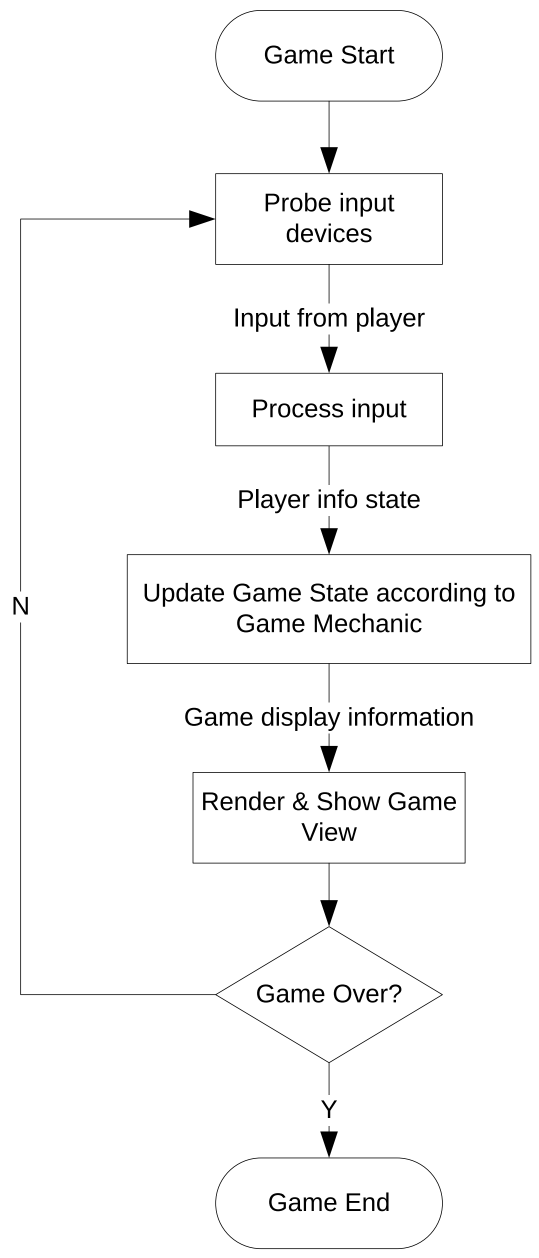

Figure 2.2: Game loop of a single player game

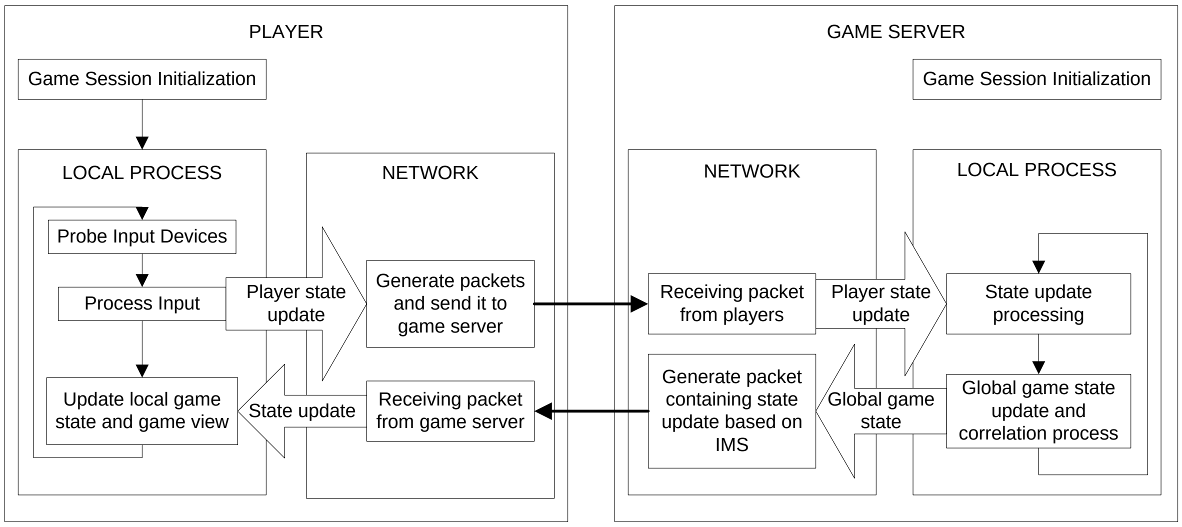
PLAYER
Game Session Initialization
LOCAL PROCESS
Probe Input Devices
Process Input
Update local game state and game view
NETWORK
Player state update
Generate packets and send it to game server
State update
Receiving packet from game server
GAME SERVER
Game Session Initialization
NETWORK
Receiving packet from players
Player state update
Generate packet containing state update based on IMS
Global game state
LOCAL PROCESS
State update processing
Global game state update and correlation process

Figure 2.3: Basic Multiplayer Game Process Loop

- Events generated by the player.
- Information of the game objects generated by the player.
- Information of the game objects modified by the player.
- Events generated by game mechanics.
- Other information not directly related to the game (e.g. advertisement by game service provider).

### 2.1.1 Multiplayer Game Mechanics

Multiplayer Game (MG) uses the same mechanism and spread it into several game machines. The basics remain the same with the difference that the game state is shared by several players at once. Figure 2.3 shows the general mechanism of an MG using Client-Server architecture.

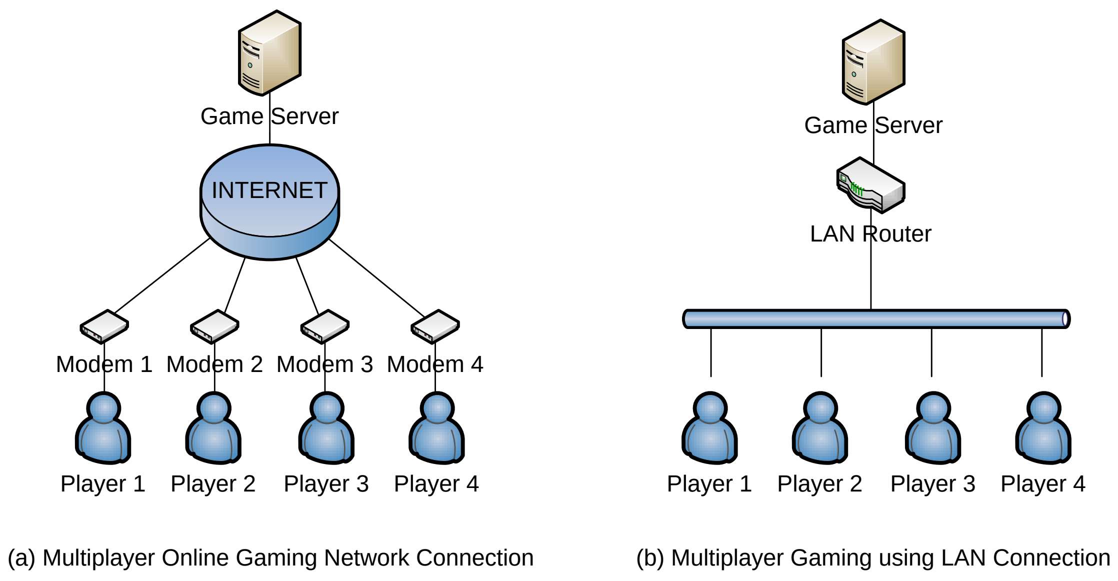

Figure 2.4: Connection Type of Multiplayer Games

The difference between SPG and MG can be seen in player input and global game state update processes. As the game states are shared by more than one player, it is very important that all players perceive the same game state. In order to do this, all players need to inform their action to the others by exchanging game state update. The first generation of MGs use cables to communicate with other players, usually using Local Area Network (LAN). Meanwhile, modern MGs are using the Internet, giving birth to Multiplayer Online Games (MOG). MOG is basically MG, but using Internet network for data communication between players. Figure 2.4 describes the connection topology between MG and MOG.

### 2.1.2 Data Communication in MOG

Data communication between players in an MOG is following the Internet standard by using IP communication protocols. There are two commonly used transport protocols: TCP and UDP. Currently, there are many debates over which protocol is best for MOG.

TCP provides more reliable mechanism to ensure that a packet will reach its destination. However, it is slower than UDP and requires more network resources mainly due to its reliability mechanism and the size of packet header. For comparison, the header of a TCP packet is usually around 20 bytes while the header size of a UDP packet is only 8 bytes. Moreover, UDP communication protocol is faster than TCP but providing no guarantee that the packet will reach its destination.

The need for reliability comes from the fact that MOG requires the game state to be adequately synchronized between all entities at all times during the game. Popular MMOGs such as *World of Warcraft* [94], *Everquest 2* [28] and *Lineage 2* [54] use the combination of TCP and UDP. It is usually done by classifying game state update based on its importance and time-critical level. Important packets with low time-critical level are being send using TCP while smaller packets with less importance but often having time-critical level are being send using UDP. However, more fast-paced games such as Counter Strike: Source [24] use UDP in most of the game data communication.

It is possible to calculate the estimation of how many bytes to be transferred in a second by a player. As an example, an MOG requires the player to send its game state update 10 times in a second. Each of the state update has the fixed size of 20 bytes. Then the size of game state update of a player in one second is 20*10=200 bytes. In order to optimize network resource usage, it is a basic requirement for game developer to design the game state update to have the smallest possible size [62]. This can be achieved by using various techniques, but the most basic technique is byte or bit coding. In this technique, the game uses byte or bit value to represent an input from the player thus making it possible to use 1 byte to represent several player's actions. One of the most recent researches about this subject can be found in [26].

### 2.1.3 MMOG Classifications

At this moment, the categorization of MMOG type is very vague with many different definitions. MOG which can be considered as MMOG needs to be able to support hundreds or thousands of players playing at the same time. Based on the game type, [58] categorizes MMOG based on the game type into:

- Massively Multiplayer Online Role-playing Game (MMORPG).
- Massively Multiplayer First-person Shooter (MMOFPS).
- Massively Multiplayer Online Real-time Strategy Game (MMORTSG).
- Massively Multiplayer Online Turn-based Strategy Game (MMOTBSG).
- Massively Multiplayer Online Sports Game (MMOSG).
- Massively Multiplayer Online Racing Game (MMORG).
- Massively Multiplayer Online Rhythm Game (MMORG).
- Massively Multiplayer Online Management Game (MMOMG).
- Massively Multiplayer Online Social Game (MMOSG).
- Massively Multiplayer Online Bulletin Board Game (MMOBBG).

Despite of having many different categories based on the game type, MMOGs have two major classification based on the game time-keeping mechanism. They are real-time games and turn-based games. In real-time games, the game time advances continuously without stopping regardless of what the players are doing. This type of game is often considered to be more realistic because the system itself is very similar to the time system in real life. However, real-time games generally consume more resources during its operation. With the game time advancing consistently and automatically, the demand of game calculations are constantly higher.

Turn-based games run the game by giving each player time to make decision about what action to take. One example of this kind of game is chess. In a chess game, two players take turn to move the pawns and only one player can move at one time. Because of its mechanism, the use of turn-based time system is generally limited for certain type of games. There are types of games that cannot be implemented in turn-based, e.g. First Person Shooter (FPS) games, as multiple players can shoot at the same time.

Besides real-time and turn-based, there are variations of time-keeping system that involves combination of both. One example is the battle system of Final Fantasy game series. They use a hybrid system called Active Time Battle (ATB) where each game character has its own ATB bar indicator. During the battle, ATB bar will gradually fills up and when it is full, the player can issue the command or action.

The type of MMOG is an important aspect and different type could result in different system requirements and sometimes they are affected by different issues. For example, real-time MMOG will have more problems with network latency while the same problem will have less impact to the turn-based MMOG. Currently, the most popular MMOGs are real-time based ones. For example, World of Warcraft and Lineage II. Therefore, this thesis will focus on the issues with real-time based games.

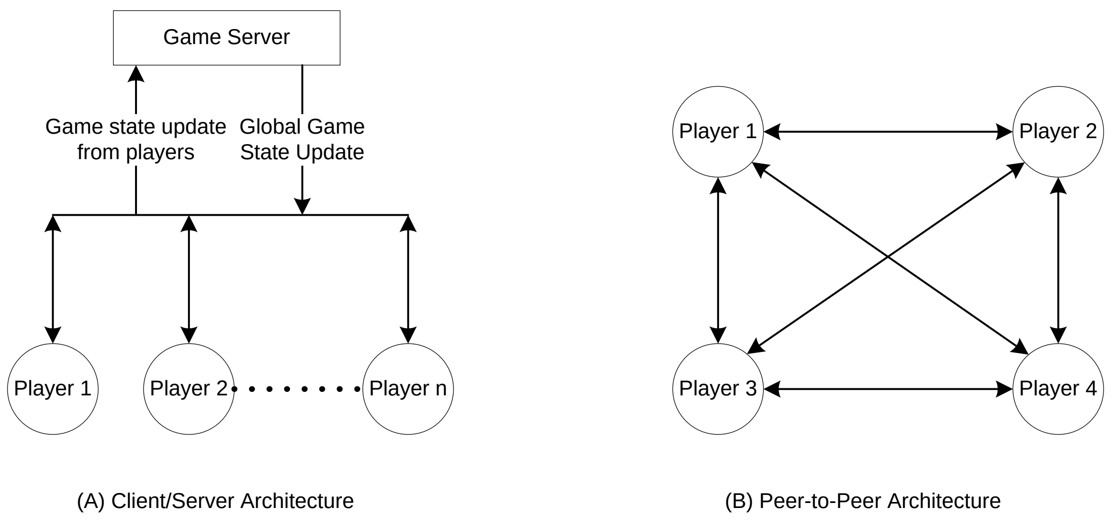

Figure 2.5: Network Architecture of MMOG

## 2.2 Networking Aspects of MMOG

Currently, there are two popular network architectures in MMOG. They are Peer-to-Peer (P2P) and Client-Server (CS) network architecture. Figure 2.5 shows the network topology of both architectures. The main difference between both architectures is the absence of a server in P2P network. CS network relies on the server to maintain the network and give other services to all clients connected to the network. Further in-depth technical reading about CS and P2P architectures and the differences between them can be found in [57].

P2P architecture is a network architecture where each member of the network contributes and shares resources with other members. There is no hierarchy level that differentiates members because they are all considered to be equal (peer). Without any server to manage the network, all members of P2P network need to maintain the network structure and resource management. In order to do this, one member is required to have network connection with several or all other members. In general, P2P and CS have advantages and disadvantages that will be discussed below.

The advantages of P2P network architecture over CS are as follow:

- There is no need for a server to maintain the network. Without server, all burden and resource requirements are shared within the members of the network.
- There is no single point of failure in the network. If there is one broken connection in a P2P network, there are other connections with others that could replace or serve as a router to perform data communication. In CS architecture, the server usually becomes the bottleneck and if the server is down, so is the whole network.
- With no single point of failure, a P2P network implementing special content addressing such as CAN [32] or Pastry [75] will be able to recover the data in the network during a major disruption.
- The nature of P2P network encourages data duplications amongst the member of the network. Therefore, when there is a major network disconnection, it is still possible to reconstruct the network and the data inside it.
- Having higher scalability compared to CS architecture. More number of members in the network means more available resources. Unlike CS where having more members means more burden for the server.

Making P2P network applicable in MMOG has been a major research field in the recent years. The main drawback of P2P network architecture lies in the scalability. Due to the fact that P2P network requires a peer to stay connected and synchronized with two or more peers, network resource usage in P2P network is individually higher than the one in CS network. For comparison, in a general P2P network, a peer will need to maintain $NumberOfPlayer - 1$ network connections to other peers and synchronize with them during the game. This operation consumes much higher network cost for a peer compared to a client in CS network. Let us consider an example of a multiplayer game with 8 players. The game requires each player to send a state update 20 times per second and in return, also receiving the state update from other players 20 times per second. If the size of one state update is 20 bytes, then the size of the state update of a player is 400 bytes per second. In P2P network, a player will need to send the state update to $8 - 1 = 7$ other players. It means, a player in P2P network will need to send $7 * 400 = 2800$ bytes per second. In CS network, a player will need to send the state update to the game server. It means, a player in CS network will only need to send 400 bytes per second. Besides sending the state update, all players in MMOG need to receive state update from other players during the game. In this case, a player in both CS and P2P networks will need to receive 2800 bytes per second. Figure 2.6 shows that the number of bytes sent by a player in P2P will increase linearly with the number of players in the game. Meanwhile, a player in CS network will have constant number of bytes sent per second regardless of the number of players.

So far, researches in P2P for MMOG have resulted in various degree of success. [3, 10, 16, 25, 29, 30, 38, 42, 44, 55, 75, 81, 82, 89, 102] have proposed solutions to solve the problem of scalability by proposing better network resource management. However, all proposed solutions generally increase the network workload for each peer by adding more operations. For example, [75] proposed a new P2P mechanism management called Pastry. Pastry is designed ingeniously to be scalable and efficient. It achieves better scalability by limiting the number of peers connected to one player. In order to maintain network structure and connect peers, it uses routing mechanism so there is no need for peers to be directly connected. Similar methods can be found in Chord [82] and CAN [32] that

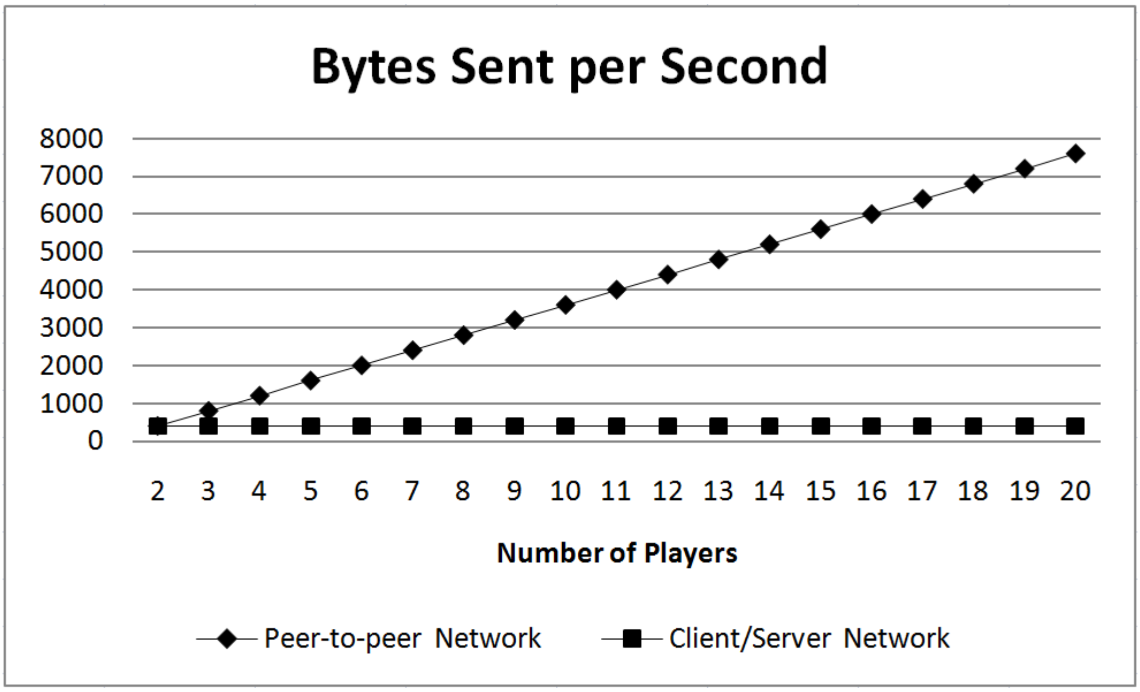

Figure 2.6: Byte sent per second for a player in P2P and CS network

proposed a solution of maintaining P2P network using Distributed Hash Table (DHT). The hash table can also be used to provide object or data addressing.

The main weakness of all solutions that attempt to solve scalability problem in P2P is the increased local resource consumption. DHT is a solid mechanism to maintain content addressing but at the same time takes memory or disk space to store. Also, computation resources will be consumed while performing any operations related to content addressing or peer routing. Therefore, it does not change the fact that P2P networks in MMOG will always have higher resource consumption overall compared to CS networks.

Solutions offered by [3, 25, 55] implement Interest Management System (IMS) to selectively pick which state update will be send to which players. In a sense, the system is very similar with the IMS implementation in CS network. Although the methods actually lower network resource usage, they also reduce the structural reliability of generic P2P networks. Because one peer will connect to less number of other peers, there is a risk of network failure when enough number of peers are having network disruptions.

Another critical issue in implementing P2P in MMOG is the network security. With the nature of P2P where all peer are considered to be equal, it means each peer will have the rights to access or modify important game state. This is also currently an active research area. Most of the proposed solutions such as [32, 47, 49] utilize selected peers that have higher position than other peers. These peers are called *Super Peers* and they have the tasks to monitor other peers. In our opinion, this kind of solution actually change the P2P network into hybrid network architecture and thus, can no longer considered as P2P network.

Super peers that have higher positions than other peers could perform network administrative tasks could become the biggest threat to the MMOG. In order to suppress the cheating by super peers, cooperation by other peers are required to monitor super peer at all time. When a certain number of peer agree that the super peer can no longer be trusted, they can impeach the super peer and promote another peer to replace the super peer. However, this solution also has severe limitations. The limitations are as follows: there is no way to detect the cheater in a swift manner. By the time everyone agrees that a super peer needs to be replaced, the damage has already done to the game state. In this case, there is no way to guarantee the validity of the global game state.

There are also solutions to deter cheating in P2P network such as the one proposed by [49]. The solutions limit the access to actual game data and instead, only transmitting player's action during game state synchronization. They argue that their method could reduce the cheating attempts. However, the biggest drawback of such system is that it does not change the fact that every peer can still load a special software locally to actually change their game stats. Furthermore, this method can be applied only to certain type of games such as MMORPG and each implementation will be unique which requires additional programming hours to the game development.

Meanwhile, the advantage of CS network architecture over P2P are:

- The server is the central point of the network. Besides the disadvantages, having server means better network monitoring and security. Since the server holds higher authority than clients, any potential problems by the clients can be dealt with quicker and it is possible to kick clients doing malicious acts out from the network.
- Higher level of authority over the clients give the server ability to determine which client can connect to the network.
- The server can help clients with low resources by performing complex operations and send the result to the clients or by simply storing data that the clients can retrieve later.

Currently, game service providers prefer to implement CS due to its business proposition. In CS architecture, it is easier for game service provider to implement 'pay-to-play'game billing system. In fact, up to now the game *Age of Empires III* (AoE III) [2] is the only popular MOG that provides the option to use P2P architecture for its online game mode. Furthermore, one main problem associated with the use of P2P in the game AoE III is bad experience when one or more of the players connected in the game have poor internet connection [33]. Another disadvantage mentioned above that motivates game developer to use CS architecture is game security. The Steam game engine that has been used to create popular MOG such as *Half-Life 2* [88] and *CounterStrike: Source* [24] mentioned in its user documentation that the decision of using CS architecture are mainly due to game data security and cheat prevention considerations [53]. For all of these reasons, this thesis will only focus on CS network architecture.

## 2.3 Current Issues in MMOG

Growing number of players, increasing complexity and variety of game contents give major issues in scalability and playability of MMOG. There are real cases in MMOG such as *Aion Online* (AO) [4], which was a highly anticipated MMORPG. During the first few months of launching, AO had major problems with scalability. The number of players who subscribed to the game was too high for what their server can handle [67]. As a result, many players complained of being rejected by the server and had to wait for a long time until they can play the game.

Scalability and playability are two of the most common measurements to assess an MMOG [5]. In general, *scalability* is the term used to describe and measure the ability of an online game system to handle or to be ready with growing number of players, game object or complexity. Many MMOGs failed to maintain their scalability and thus cannot provide appropriate services to the players. One example is the MMORPG Aion Online [4] that was having scalability issues due to the unpredicted number of players playing at the same time during the first few months of its release [67]. In this dissertation, the research scope is to tackle the issues of scalability related to the networking.

*Playability* is the term used to describe quality of the gameplay of an MMOG. Factors that define playability are different for each type of games. In general, playability is comprised of game balance, game user interface, display quality and all other stuff that affect player's experience while playing the game. As an example, storyline in MMORPG holds bigger effect in playability compared to MMORTS. In this dissertation, the research scope of playability is to solve the problem caused by cheating in MMOG. Cheating disrupts playability since cheating usually performed to give the cheater advantage over other players or the game mechanic itself.

### 2.3.1 Networking Issues

Since this thesis focuses on CS network, all of the networking issues discussed in this chapter are the entire issues encountered CS network. Figure 2.7 depicts the network connection and communication between players and game server. As explained before, the most basic method for game state synchronization is by making the players to send its game state update periodically while the game server will receive the game state update from all players, update the global game state and send the global game state back to all players. However, this is not the most efficient method to synchronize game state. There are scalability issues related to the networking when using the basic method. This section will discuss about those issues, current solutions adopted by game developers and the most current solutions proposed by other research.

#### 2.3.1.1 Network Latency and Packet Loss

The most common problems with the Internet these days are network latency and packet loss [65]. Network latency is a measurement of how long a packet will arrive to its destination. In an ideal world, a packet will arrive instantly to its destination. Nevertheless, network latency always exists everywhere in the Internet regardless of Internet bandwidth or throughput.

All Internet applications including MMOG have a certain tolerance to network latency.

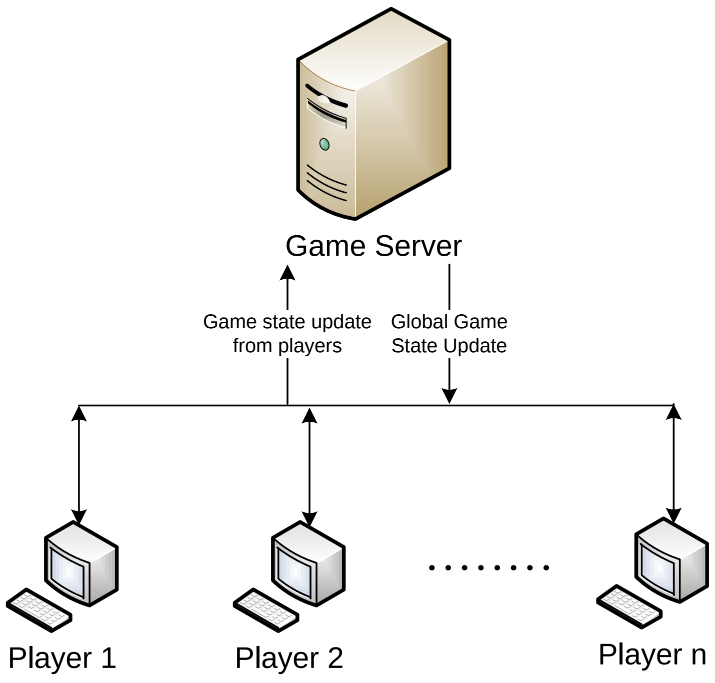

Figure 2.7: Network Communication Between Players and Game Server

However, there are times where the network latency are too high and in MMOG, the late arrival of game state update could disrupt the whole game state synchronization. Depending on the game type, certain MMOG could tolerate network latency better than others. In this case, usually turn-based MMOG would perform better than real-time MMOG in the Internet with high network latency. The reason is because the mechanism in turn-based MMOG forces a player to wait until the other player finishes its turn.

In Real-time MMOG, game state update communication is performed constantly. Because of the mechanism, one state update will quickly lose its importance if not delivered in timely manner especially with fast-paced games. In this case, network latency or packet loss are the biggest problem in game state synchronization in MMOG. The effect of network latency or packet loss can be seen visually in MMOG. During high latency or packet loss, the game view of a player will become inconsistent with other players. One common sign of network latency that can be recognized is when the movement of other players seems jittery. This happens because during high latency, a player cannot properly receive state updates of other players. As a result, the local game system in the player's machine does not have enough data to render the movement of other players properly. We will discuss this topic further in Chapter 5.

Another contributing factor to network latency problem is that each entity in the MMOG could experience different latency in network communication. The network latency occurring in one player could be higher than the other player. This condition leads to the problem where game server will receive game state update from players in different time. As explained in the previous section, the game server will need to perform global state update every tick using game state updates received from all players. When one of the players is having problem with network latency or packet loss, the game state update from that player could arrive too late to be included in the global state update operation. As a result, the most recent game state stored and distributed by the game server is no longer properly synchronized and the same condition applies to all players who receive the game state update from the game server. Therefore, fairness of the gameplay could be compromised and players with better Internet connection will have a definite advantage over other players with high network latency.

One real life example of this problem is the multiplayer mode of the game *Armored Core 4: For Answer* [7]. The writer himself experienced the effect of high network latency during multiplayer game session with players from Japan. Since the game server is located in US and Japan, Australian players is very susceptible to high latency. It is quite common that players in Japan are often use this condition to their advantage by using fast-movement to approach and deliver the killing shot to the other players with high latency. What the players with the latency can see is their enemy suddenly disappear and appearing from one spot to the other. Also, the latency could freeze the players' action thus leaving them no chance to retaliate or performing evasive maneuver. Investigations by [70] further describes the impact of network latency in real-time MG.

The solution commonly applied by game service provider is by providing more bandwidth and implementing newer Internet technology that uses better communication protocols. This solution is the quickest method to improve stability but also the most expensive one. Also, this solution does not solve the problem if the cause of the latency is from one of the player's Internet connection. Therefore, the solution is not viable to be implemented in smaller MMOG and in the condition where the players' Internet connections have high latency.

#### 2.3.1.2 Dead Reckoning

A widely implemented method by game developer to compensate high network latency and packet loss is Dead Reckoning (DR). DR method works by providing the guesstimate off current location of a player [53] and thus can be applied only in certain types of MMOG. In a way, DR works by increasing the tolerance of an MMOG to network latency. DR is a popular and essential method in MMOG programming because approximately, up to 70% of network resource usage in MMOG comes from sending or receiving player's movement data [5]. DR have two types of operation, they are Interpolation and Extrapolation.

Interpolation is used to fill the gap between state update to make the game animation smoother. As shown in Figure 2.8, player's machine will produce predicted game state of a player in order to render the animation during the game. Without interpolation, the animation will appear as jumpy or jittery. High quality animation requires the standard

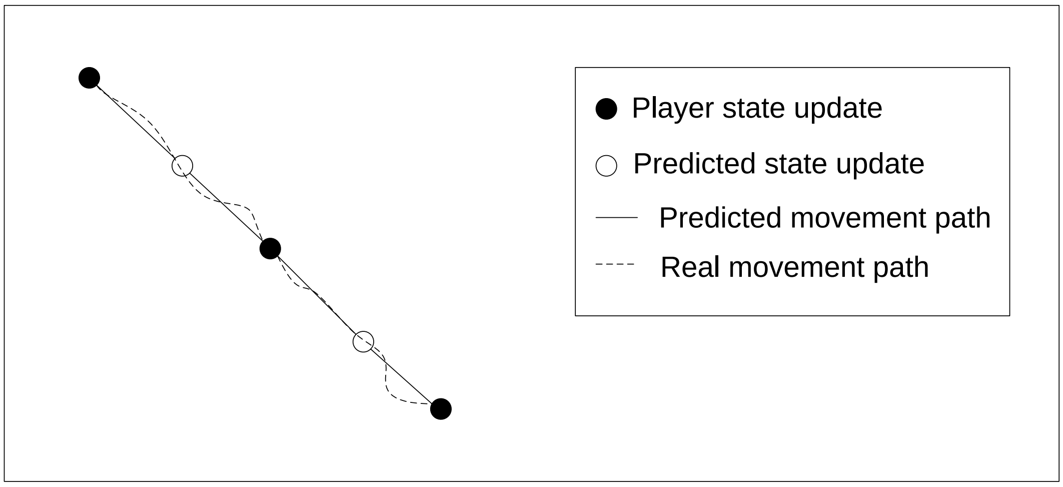

Figure 2.8: Game state Interpolation

60 frames to be displayed in a second. If the same standard is applied in the frequency of game state update, it will consume too much bandwidth. Therefore, even without high network latency, periodic state update is often not enough to produce smooth game animation. The simplest method of interpolation is to draw a line between state update and animate the player so it will appear that the player moves by following the line path. The problem here is that some games tend to have more curvy movement pattern. FPS games are good examples where all players usually walk freely in the game world. Thus creating non-straight movement pattern most of the times in the game. However, other players can tolerate minor deviation of interpolation and the interpolation result usually does not affect the game result.

Extrapolation is used to provide prediction during high network latency. During that time, the game cannot receive several state updates and as a result, need to come up with a dummy location of a player in order to render the game smoothly. Figure 2.9 describes a scenario where the game calculates two dummy state update using extrapolation. There are various methods to perform extrapolation calculation. Formula 2.1 is one example to

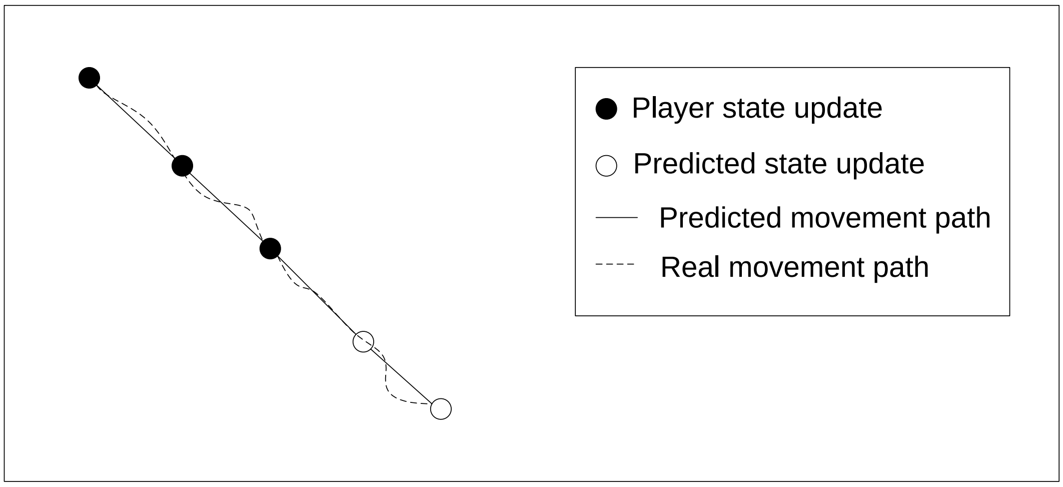

Figure 2.9: Game state Extrapolation

get the distance travelled by a player ($D$) based on the speed ($v$) and time ($t$).

$$D = vt \tag{2.1}$$

In general, the fundamental background of most of the research about DR in multiplayer games comes from the effect of network latency or packet loss. Research and discussions about it can be found in [104, 104, 27, 78]. While some of the research [31, 22, 78, 12] conclude that network latency does not have significant impacts in player behavior and game result, there are others [101, 53] that conclude the opposite. Interestingly, those with different conclusions observe different multiplayer games. At this stage we assume that the magnitude of the impact of network latency in multiplayer online games depends on various different aspects such as type of players, type of games and network conditions. Hence, there are certain situations where DR could improve gaming experience.

There are other related work about other methods of dead reckoning in multiplayer games such as in [71, 84, 69, 50]. Some of them compare their work with DR method described in IEEE 1278 [43] which is the standard of application protocol in Distributed

Interactive Simulation (DIS). The standard formula to perform Dead Reckoning calculation is as follows [8]:

$$X_1 = X_0 + v_0 \Delta t + \frac{1}{2} a \Delta t^2 \tag{2.2}$$

Where $X_0$ is the initial location, $X_1$ is the predicted location, $v_0$ is the initial speed and $a$ is the acceleration and $\Delta t$ is the time difference between $X_0$ and $X_1$. In the end, this formula becomes the basis of the IEEE standard formula for dead reckoning. In practice, IEEE standard for distributed interactive simulation (IEEE Std 1278.1-1995) [43] is often adopted as DR method in video games even before MMOG was born. The method implements multi-step DR calculations, as described in [84]. The formulas for the first step are:

$$x_t = x_{t'} + v_{t'} \Delta t \tag{2.3}$$

$$x_t = x_{t'} + v_{t'} \Delta t + \frac{1}{2} a_{t'} (\Delta t)^2 \tag{2.4}$$

Where Equations 2.3 and 2.4 are the first order and second order respectively. Meanwhile, the first and second order equations for the two-step DR are:

$$x_t = x_{t'} + \frac{x_{t'} - x_{t''}}{t' - t''} \Delta t \tag{2.5}$$

$$x_t = x_{t'} + v_{t'} \Delta t + \frac{1}{2} \frac{v_{t'} - v_{t''}}{t' - t''} (\Delta t)^2 \tag{2.6}$$

Where Equations 2.5 and 2.6 are the first order and second order respectively. In all formulas, $x_{t'}$, $a_{t'}$ and $v_{t'}$ are the most current position, acceleration and velocity of a player. The current position means the position received in the last known game state update. Meanwhile, $x_{t''}$, $a_{t''}$ and $v_{t''}$ are the position, acceleration and velocity of a player obtained from the second last known of game state update. Also, $\Delta t = t - t'$ for all formulas.

The equations above gives a good result to predict player's position with some exceptions. Two-step formula, despite of including more variable, tend to have lower accuracy. Also, Two-Step formulas requires more computation power to calculate the result. Another fatal flaw is that such Formula could only predict a linear movement pattern with minimum changes in angular velocity and acceleration [69]. In practice, player's movement pattern is hardly linear all the time. Therefore, current DR methods produce errors quickly in the game and current remedy by game developers are by increasing the frequency of game state updates. There are also attempts to improve DR using other methods such as the one described by [50]. However, their proposed solution has limitations because of its high complexity. Furthermore, the solution also suffers from the same drawback as the standard DR method. It could only provide accurate approximation when there is minimum variance in player's angular velocity and acceleration. Thus, limiting the implementation of solution to other type of games such as racing games or first person shooter.

One of the latest research works about DR in online games utilizes Artificial Neural Network (ANN) [59] in order to learn the movement pattern of specific games and provides the estimation of player's location based on the pattern learned by the ANN. However, this research lacks the specific use and analysis of its usage in 3D games. Also, the research itself uses the simplest form of ANN, which lacks the ability to adapt with constant change of movement patterns.

### 2.3.2 Network Resource Management

One practical way to improve scalability in MMOG is by providing better network resource management. Using the same amount of network resources, it is possible to have better scalability by increasing the efficiency of network resource management. There are many ways to reduce the effect of latency and increase scalability [104, 105, 15, 64, 9, 63, 100, 66] and one of them is by implementing Interest Management System.

#### 2.3.2.1 Interest Management System

Interest Management System (IMS) is a management system implemented in MMOG to improve the efficiency of network resource usage by sending parts of game state update selectively to players according to certain categories. The conventional way to synchronize game state updates where a player will receive the global game state from the game server. IMS use the categorization mechanism to determine which game state update is relevant for which player. Therefore, a player no longer needs to receive the full size of game state, but only the parts relevant to the players. There are plenty of ways to design the categories based on the game types or just based on the preference of game developers [13, 14, 90].

One of the most commonly implemented is by using Area of Interest (AOI) [25, 3, 84]. Area of Interest works in a way that every player will have a virtual circle with the player as the center. Anything that are in the circle are included in game state update that the player will receive. The simplicity of this method makes it a popular method but also at the same time, having high resource consumption. The high resource consumption is because the player need to perform AOI check periodically. Because the game server stores all of game information, the task of performing AOI check must be done by the game server. For players, this is a good thing. For game service provider, this is a disadvantage. MMOG usually have hundreds or thousands of players connected at the same time. If the game server needs to perform AOI check for each player, the resource cost to do it would put a heavy strain on the game service provider.

Another practical method for IMS is by making shards of the game world. *Dungeon Siege II* is one example of the game world division. The game world in Dungeon Siege II are comprised of many smaller maps with specific points at the edge of the map acting as exit/entry points. While this method is very effective, game development cost would be higher and the game would take longer to develop because each segment of the game map needs to be customized so every shard will look and feel natural.

#### 2.3.2.2 Network Workload Evaluation

By default, it is always possible to improve scalability by providing more game server and more bandwidth to the network infrastructure. However, this method is not cost-efficient and does not provide long-term solution. The article [87] explains that the most expensive part to maintain the network in MMOG is the bandwidth required by the game to have thousands of players connected at one time.

Smooth game market-launch is also a vital part to the success of an MMOG. The challenge here is to predict and provide adequate network resources for an MMOG. Over-estimation will cost the game service provider while underestimation of network resource requirement will cause scalability and playability problems. The game *Anarchy Online* and *World War II Online* are perfect examples where they had server and game stability problems during launching, and thus affecting the game negatively.

Unfortunately, the only common method for game service provider to predict network resource requirement for game server is by using simple estimation such as:

$$N = n\theta \tag{2.7}$$

where $N$ is the network workload of a game server, $n$ is the number of players and $\theta$ is the number of network workload for each individual player. To determine $\theta$, game service provider can use estimation based on the size of network packet for game state update of the MMOG.

The proposed solutions by other research such as [85, 103, 76, 17] focuses on the network traffic and the result are collected from real server or simulated game sessions. There is one research potential in this area. Other research tend to use different metrics and standard. It is possible to create a mechanism to evaluate network workload. Thus, creating better comparison and analysis for different methods and solutions proposed by others.

### 2.3.3 Cheating Issues

Cheating is one of the major problems in MMOG. Cheating is an act where players or other third parties attempt to modify the result of the valid game mechanics. The modification could be implemented to player's game state, game server's global game state or even the binary program of the MMOG itself. The result of cheating always affect gameplay experience. Thus, the aim of cheating is almost always to provide advantages of the cheater over other players.

As a result, cheating could deter significant number of honest players to play an MMOG. Therefore, cheating is one of the major concerns for game service provider as well. There are plenty of ways to cheat in MMOGs, a good review about cheating in networked games [92] classifies the cheat into several categories (Table 2.1).

| Level | Cheat Type |
|---|---|
| Game Level | Bug<br>Real Money Transaction (RMT) |
| Application Level | Information exposure, Invalid Commands<br>Bot/Reflex Enhancers |
| Protocol Level | Suppressed update, Timestamp, Fixed delay, Inconsistency<br>Collusion<br>Spoofing, replay<br>Undo<br>Blind Opponent |
| Infrastructure Level | Information Exposure<br>Proxy/Reflex Enhancers |

Table 2.1: Cheat Categorization by [92]

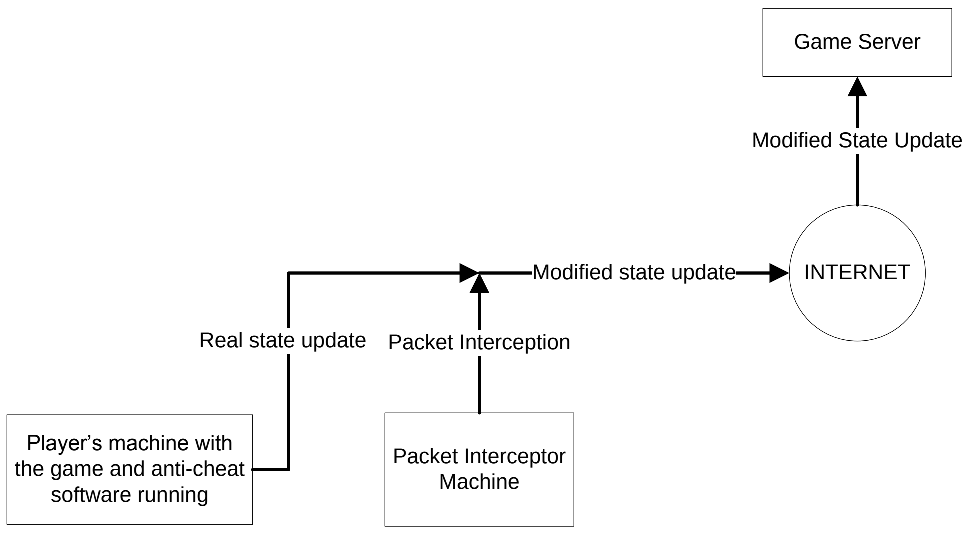

Figure 2.10: Packet Interception Cheat

From all of the classification of cheating listed in Table 2.1, Bot is one of the cheating types that currently has no solutions in CS architecture. Bot is difficult to detect because it does not work against the game mechanics. Instead, it helps players to do certain tasks in the game. Currently, game developers and game service providers rely on the anti-cheat software running in player's machine to detect the presence of bot program in the machine's memory. However, it is possible to evade the detection by anti-cheat software by using the method depicted in Figure 2.10.

In Figure 2.10, the player runs the game and anti-cheat applications in the machine. At this stage, anti-cheat software does not detect anything illegal in the player's machine and allow the player to join and play the game. However, there is another machine intercepting and modifying the real state updates into the ones generated by bot. With this method, the game server will assume that everything is fine because the anti-cheat software is running in player's machine and it reports that there is no bot program running in the player's machine.

Current research to detect bot generally implement CAPTCHA [37, 99], network traffic analysis [18, 91], trajectory analysis [20], behavior analysis [86] and bot-detection program [77].

CAPTCHA works is by prompting the player with a set of characters and it requires the player to type the exact characters into the input field to prove that the player is truly a human player. Based on its basic mechanic, CAPTCHA has the limitation that it will cause annoyance to the players playing fast-paced real-time games [37]. Another weakness of CAPTCHA is that it is possible to circumvent CAPTCHA by using the automatic character recognition. Online service such as [11] is one example of available means to eliminate human intervention to beat bot detection using CAPTCHA.

Network traffic, trajectory and behavior analysis rely on statistical analysis to recognize certain pattern. However, as soon as the bot is programmed to include randomness, these methods would most likely fail to detect the bot. It is possible to re-design the formula in order to recognize the random factor introduced by bot, but this method is not cost-effective because it takes only one or several lines of bot code to change the random pattern. Therefore, it is a good research field at the moment and thus we make it into one of our research objectives in this thesis.

# Chapter 3

# Uniform-Shaped Game World Partition System

## 3.1 Introduction

Generally, game world partitioning can be categorized as a part of Interest Management (IM). IM is widely used to reduce the amount of network communication by sending the data only to recipients selected according to specific criteria(s) [61]. Game world partition divides game world into smaller cells which cover the whole game world and it means each cell contains a small part of the game world. As the game state updates are transferred through the network in forms of packets, game world partition significantly reduces the overall number of packet transferred during game session. The reason is that a player will send state updates only to other players located in the same cell and not to every player in whole game world. An additional reference regarding the advantages of game world partition can be found in [13].

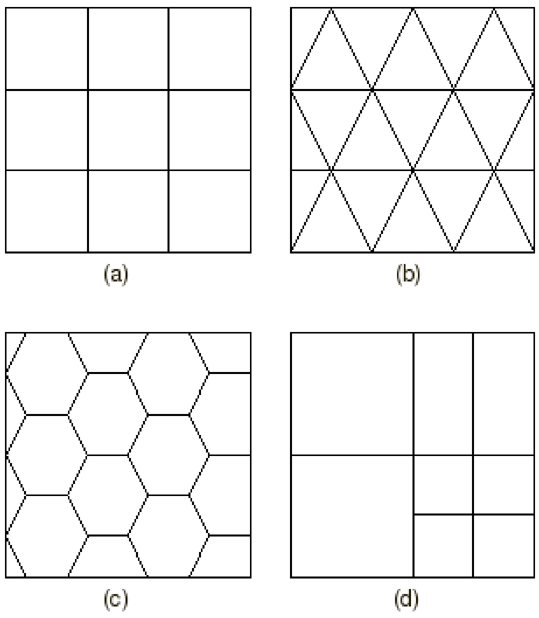

Figure 3.1: Various game world partition methods: (a)Rectangular (b)Triangle (c)Hexagonal (d)Variable Rectangular

Figure 3.1 shows various methods to divide game world into cells. These methods can be divided into two major categories: uniform and variable partition shapes. Uniform partition divides game world into cells with same shape and size. The most commonly used shapes used for uniform partition are: square/rectangular, triangle and hexagon. Meanwhile, variable partition creates cells with different shape and size. There are many methods and algorithm of how to make partition such as the ones described in [56, 13]. Game world partition is suitable to MMOG because it could reduce the effect of network latency by limiting the number of the recipient of state update. Another reason is the localization of game state inconsistency when network latency occurs as it will only affect one or some small portions of the whole game world. However, this method still cannot cope with the major network latency such as the ones that affect one whole country.

## 3.2 Discussion

Despite of the advantages of game world partition, player's game machine usually have limited available computational, memory and networking resources. Some of the partition methods that requires complex operation is not suitable as the player's game machine need every available resource to process game mechanics [6]. That is the main reasons of why our research only involves simple and common partition shapes. Therefore, our research questions for this chapter are:

- What is the metric to measure the performance of one partition shape?
- What methodology can be used to perform the comparison?
- How do partition shapes perform compared to each other?
- How to improve the performance while keeping simplicity and efficiency? The definitions of simple and efficient here are determined by the trade-off between game machine resources consumed by game world partition system and the improvement in network resource consumptions.

Before we answer those questions, due to many possible variations in real games we are limiting our research scope and general concept by using definitions described in Table 3.1. In association with definition in Table 3.1, during a CM (Cell Migration) process, data related to the migrating player will need to be transferred to the new cell and all members of old and new cells need to be notified of new position of the migrating player. In order to provide smooth transition, it is necessary to transfer data even before CM event occurs which is CPM (Cell Pre-Migration) event. While CM process is triggered when a player moves to different cell, CPM is initiated when a player's location is in certain range

Table 3.1: Definition of terms

| Terms | Definition |
|---|---|
| Point | Smallest region in the game world pointed by game coordinate system and may contain a player. |
| State update | Periodic process to send player or object information to other entities in the game in order to keep game data consistency. |
| Cell Migration (CM) | Events when a player moves to another cell. |
| Cell Pre-Migration (CPM) | Events when AOI of a player crosses a cell border. |
| Area of Interest (AOI) | A virtual domain surrounding a player where the associated player will receive all data about entities or objects inside it with higher priority. |
| Line of Sight (LOS) | Areas which are visible to the player. |

to the border of a neighboring cell. Each player has an Area of Interest (AOI) where it forms a virtual domain surrounding a player and all entities inside AOI radius will receive state update from the player. When AOI of a player crosses the boundary of a cell, CPM process will start and it will end when the cell border is no longer inside the AOI. After CPM process has stopped, a player can purge the data involved in CPM that is no longer used in order to save memory resources.

Derived from our definition of CM and CPM, we can assume that those events consume more network resources than normal periodic state update. Because of this reason, we are using the rate of CM and CPM as one of the metrics to analyze performance of different partition shapes. We measure the rate by counting how many CM and CPM occurred

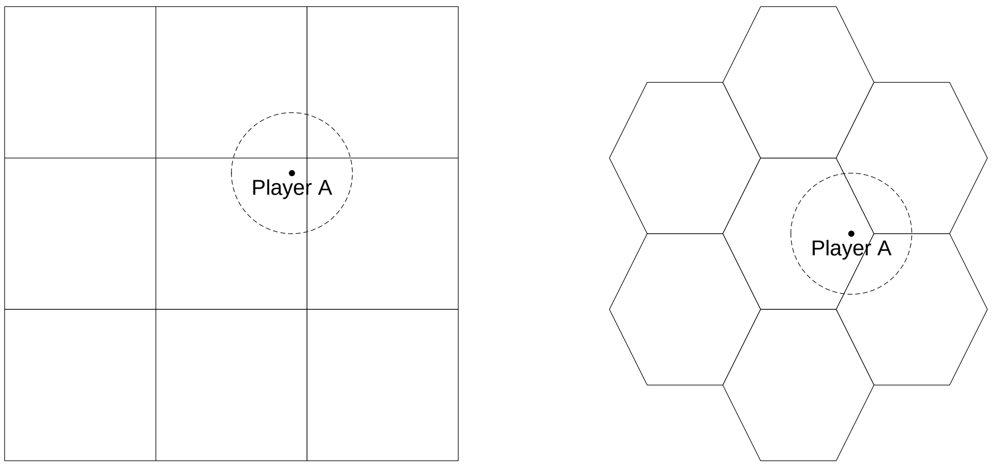

Figure 3.2: Player AOI in different partition shapes

during simulated game sessions. Figure 3.2 illustrates a player where its AOI crosses the border of its home cell. Based on its geometry characteristic, the total number of neighboring cell is different for each partition methods. In the worst case, a player in square or rectangular cell will have maximum three neighbours while hexagonal cell will have maximum two neighbours. During CPM, additional neighboring cell means more state update to be sent over the network which will affect network resource usage directly. The next metrics are related to CM and CPM where its occurrence follows one another. During game session, a player can move freely in the game world and apart from game world topology, a player can move with any kind of pattern. Since all of these game world partition and mechanism are invisible to player, a player might trigger CM and CPM repeatedly as illustrated in figure 3.3.

We call these events Consecutive Cell Migration (CCM) and Consecutive Cell Pre-Migration (CCPM). In summary, metrics definition to compare the performance of different partition methods are: $\eta$ - Cell Pre-Migration rate, $\theta$ - Consecutive Cell Pre-Migration rate, $\nu$ - Cell Migration rate and $\mu$ - Consecutive Cell Migration rate.

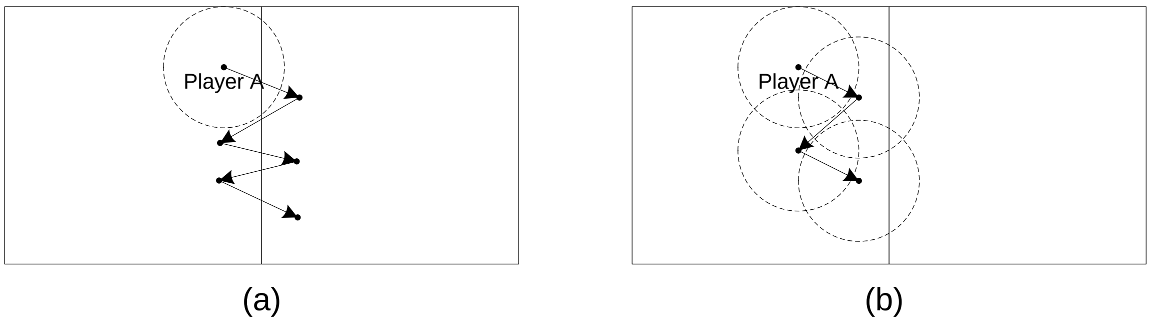

Figure 3.3: Consecutive Cell Migration (a) and Consecutive Cell Pre-migration (b)

Additionally, we consider the number of neighbours during CPM as one of the metrics as well. The reason is because although one partition shape achieve a lower number of CM, CPM, CCM or CCPM, the same result might not be achieved by using other type of games.

### 3.2.1 Mechanism

In order to get data of the events mentioned in the previous section, we developed a game world and player simulator. Considerations of using simulation as our methodology are based on: flexibility to include and observe various parameters, full control of simulated environment and possible expansion for future research or experiment using additional data. In order to have a fair comparison between partition methods, we used the same player movement pattern and same system parameters (eg: game world size, number of player, radius of AOI). Additionally, we used cell area to define the size of a cell. More information about simulator parameters can be found in the next subsection.

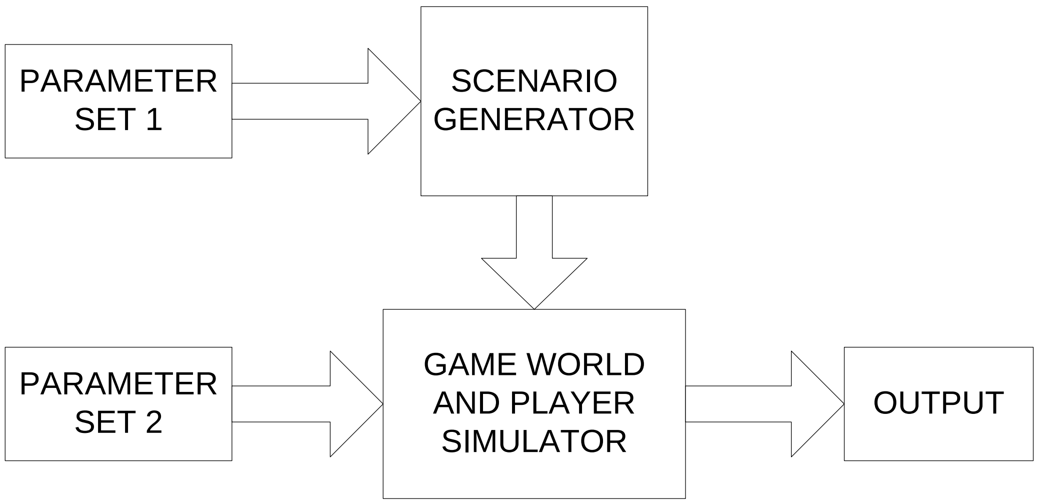

Figure 3.4: General structure of Game World Partition system simulator

### 3.2.2 Simulator design

Our simulator consists of two main modules with different parameter sets (Figure 3.4), they are: Scenario Generator and Game World and Player Simulator. At current stage, we used Scenario Generator to generate players' movement pattern with random direction and speed. These randomized player movement patterns can be exchanged with real data from real games as it becomes available in the future. Game World and Player Simulator module will take the result from Scenario Generator and combined with parameter set 2, will simulate a game session. At the end of simulation, a final report will be generated and it contains the final value of $\eta$, $\theta$, $\nu$, and $\mu$ for all partition shapes.

Figure 3.4 also shows that each main module has its own parameter set. Parameters for Scenario Generator are:

- $P$ - Total number of player

  Total number of player in the game world. It is a variable initialized before simulation, but it is not possible to change this parameter in the middle of simulation.

- $L/H$ - The width/height of game world

  The game world is represented as a square which has uniform width and height in pixels.

- $t$ - Simulation duration and $\alpha$ - Length of one game cycle

  We use seconds as simulation duration and $\alpha$ represents the time of 1 cycle in game session. In our simulation, we assume that 1 cycle is 100 ms because there is a general consensus in game programming that when a cycle is more than 150ms, the players can perceive it as game lag. This value is uniform across players where their new location is generated every 1 cycle based on its movement speed and direction.

- $v$ - Player movement speed

  Player movement speed represents how fast a player can move across game world. In practice, player movement speed in different type of game varies greatly. In our simulation, we use scale 1-9 with 1 is the slowest and 9 is the fastest in order to represent different type of game. One important thing to note is, this parameter indicates the maximum movement speed of all players and movement speed itself is not always constant during simulation. Furthermore, we applied 360 degrees of possible movement direction for each player. Randomized movement pattern is achieved by using random speed, direction and duration.

Meanwhile, the parameters for Game world and Player Simulator are:

- $A$ - Cell Size

  Cell size indicates the area size of a cell. In order to emphasize fairness between different partition types, cell size is defined by the area of a cell. This is to ensure that different partition type has the same amount of data transferred for migration process. Our assumption is that cell with the same area size will have the same network resource requirement to transfer its information state to a player. To achieve this, we are using cyclic polygon as the basis of cell's shape. Commonly, we have three main shapes: rectangle, hexagon and triangle. Given the area of a polygon and the number of vertices, we define the value of the circumradius of a cell using formula 3.1.

$$R = \sqrt{\frac{2A}{N \sin(\frac{2\pi}{N})}} \tag{3.1}$$

where:

$R$ = Circumcircle radius of a cell

$A$ = Cell area

$N$ = Number of sides

With the value of R, we can calculate the coordinate of each polygon vertices as illustrated in figure 5 with the same formula used to find coordinates of cyclic polygon.

- $\gamma$ - Radius of Area of Interest

  Area of Interest indicates the virtual domain of a player. In our simulator, we increase the value of $\eta$ when the AOI crosses boundary of another cell and $\theta$ when it crosses the same cell again after previous CPM happened in less than two seconds. The same threshold applies to $\mu$ where it will increase when a player returns to its old cell in less than two seconds after migration.

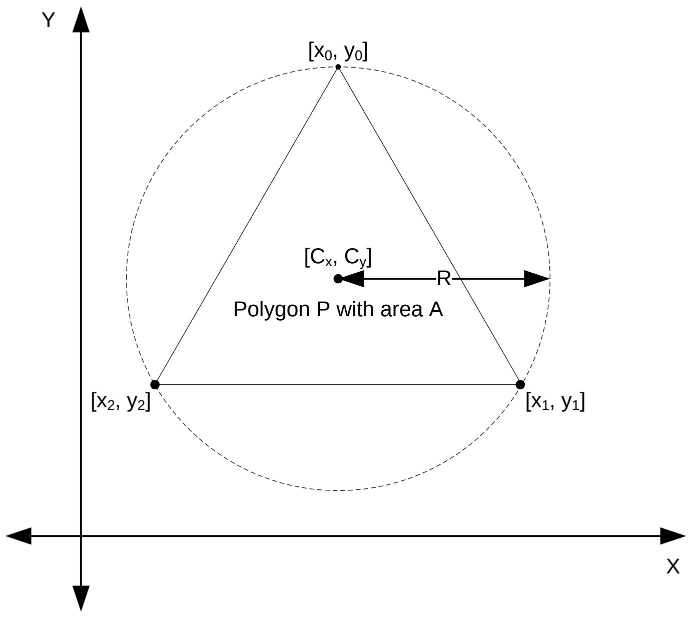

Figure 3.5: Circumradius of a triangle

### 3.2.3 Modified Brickworks Partition Method

During our initial simulation with 4 basic partition shapes, we found that there are differences in $\theta, \eta, \nu$, and $\mu$ for each partition method. Backed up with our initial hypothesis and analysis of current network limitations, we proposed a partition method that is simple to implement and could generally perform better than other basic partition methods in most cases. Firstly, with limited resources in game environment, we decided to use square as the cell shape because it is easy to implement and does not need high resource for in-game operations such as edge or border detection and coordinate calculation. Secondly, hexagon shape has the least number of possible neighbour and therefore quite popular in practice. Using square shapes and regular tiling pattern, we have three possible neighbors as the worst case while the hexagon has two neighbors. By changing the cell shape to square, we can achieve the same result. Additionally, square is more simple in terms of AOI edge detection compared to hexagon shaped cell. In hexagon shaped cell, calculating AOI edge detection usually involves point-to-line distance formula. Meanwhile in a square cell, it will be as simple as comparing $x$ and $y$ coordinate. Finally, in order to reduce the negative impact of high $\theta$ and $\mu$, we created virtual partition within a square cell. We call this method as *Brickworks with Internal Partition* (BIP).

The difference between BIP and 4 other basic partition shapes is in the use of fixed internal partitions. [5] suggests the implementation of duplicated and shared area between two cells. It means, a certain range of the edge of two neighboring cells is managed by two game servers. A player that enters the shared area will establish connection with the other server, thus providing smooth transition during CM. However, this method actually wastes the data storage and also has the potential of having synchronization problem because of the shared area. BIP does not have shared area, as each cell has its own game server as the owner. BIP also provides a smooth gameplay because of the use of AOI to trigger CPM. Another obvious advantage of BIP over other partition shapes is that the maximum number of possible neighbors is as low as hexagon shape, but with better implementations due to its square shape and internal partitions.

BIP begins with brickworks pattern as described in Figure 3.6(a), each of single cell in brickworks partition contains virtual partitions which divide the cell into 9 rectangles (Figure 3.6b). The mechanism is to detect location of a player in the game world. As the player moves, it will receive state update of the whole cell just like normal. However, during CPM event, the player will not receive the whole information of possible future cells. Instead, it will receive only part of it as it is crossed by player's AOI. This way, when a CCM or CCPM occurs, wasted network resource usage will be less than ordinary method. With virtual partitions, a cell will consist of: 4 small squares at the corners, 4

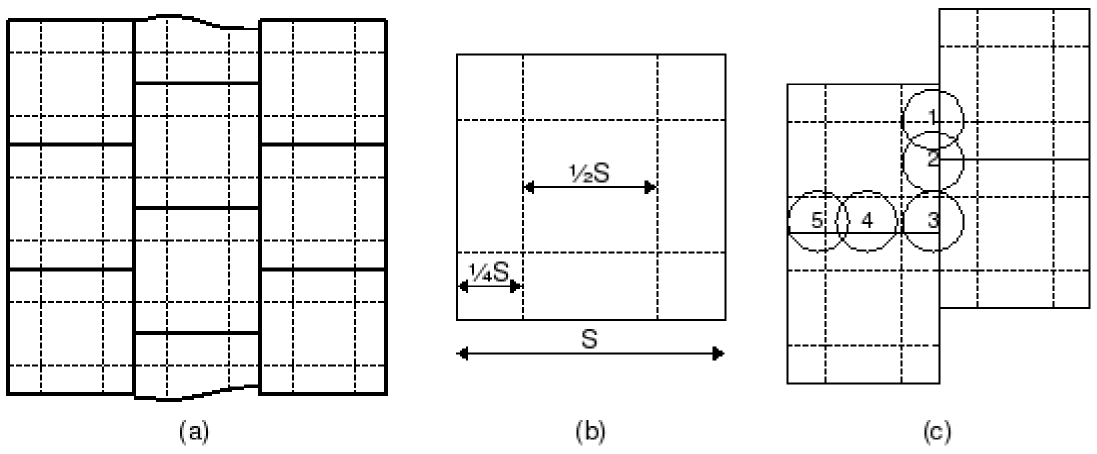

Figure 3.6: (a) Brickworks partition Layout. (b) 1-cell view. (c) Various CPM and CM scenarios

rectangles and 1 square in the middle. Assuming that the length of one side of a cell is $S$, then placement of virtual partition follows these rules: the length of the side of middle square is $\frac{1}{2}S$. For smaller square, the length is $\frac{1}{4}S$. Rectangular has $\frac{1}{4}S$ for 2 sides and $\frac{1}{2}S$ for the other sides. That makes the area of smallest squares as $\frac{1}{16}S$, $\frac{1}{4}S$ for the middle square and $\frac{1}{8}S$ for the virtual rectangle partition.

For BIP, network resource usage is lower compared to other basic shapes because there are less data to transfer during CM and CPM. More efficiency can be achieved when CCPM or CCM occurs. There are several possible scenarios for CPM and CM events in PB as illustrated in Figure 3.6(c). Throughout scenario 1 to 5 in BIP, the worst case is when the AOI of a player crosses two small squares and one rectangle (scenario 3) and $\gamma > \frac{1}{4}S$. The best case is when it crosses only 1 small squares and $\gamma < \frac{1}{4}S$. Based on possible scenarios, the condition where $\gamma \leq \frac{1}{4}S$ is preferable in order to get benefit from best case.

### 3.2.4 Cell Placement Algorithm

Our implementation of GWP requires construction and placement of cells covering the whole game world. Some multiplayer online games such as *Counter Strike Source* [24] and *Quake III Arena* [1] allow players or other third parties to design their own map. If we want to implement GWP, then there are two options. The first one is by placing the cells manually for every new map released or the second option is to do it automatically. Generally, manual cell placement results in the minimum number of cells required to cover the game world. However, the process is time consuming and it is possible to have bad results due to human-error. Therefore, we designed our own algorithm to do this task automatically. The main reason is to have a consistent cell placement pattern and in the end it will give better validity for our research result. During simulation, we found that our algorithm may help game developer who wants to implement GWP in their games; especially with the games where players or third parties can participate to create and deploy their own maps. One important thing to note is the main objective of our algorithm is not to find the best result of tile placement.

Our basic algorithm begins with making a bounding rectangle. The rectangle consists of 4 vertices and covers the whole game map (Figure 3.7(a)). There are two important vertices: bottom-left vertices $(X_{min}, Y_{min})$ and top-right vertices$(X_{max}, Y_{max})$. The next step is to put the cells in tile pattern covering the bounding rectangle as shown in Figure 3.7.b. After this step, we need to go through all cells and delete the unneeded cells that are not covering any parts of the game world. The whole process is described in Algorithm 1 and although algorithm 1 is for the square cell, other shapes also follow the same mechanism for the cell placement.

**Algorithm 1:** Automatic cell placement algorithm

```
Input: Game world map
Output: Cell placement that covers the whole game map
Get Top-left Vertices($X_{min}$,$Y_{min}$);
Get Bottom-right Vertices($X_{max}$,$Y_{max}$);
$CellIndex = 0$;
$Y_{current} = Y_{min}$;
while $Y_{current} < Y_{max}$ do
    $X_{current}$=$X_{min}$;
    while $X_{current} < X_{max}$ do
        Assign position of Cell($CellIndex$) to ($X_{current}$, $Y_{current}$);
        $CellIndex = CellIndex + 1$;
        $X_{current} = X_{current} + CellWidth$;
    end
    $Y_{current} = Y_{current} + CellHeight$;
end
foreach $Cell(i)$ do
    if $Cell(i)$ does not cover Game World then
        Delete($Cell(i)$);
    end
end
```

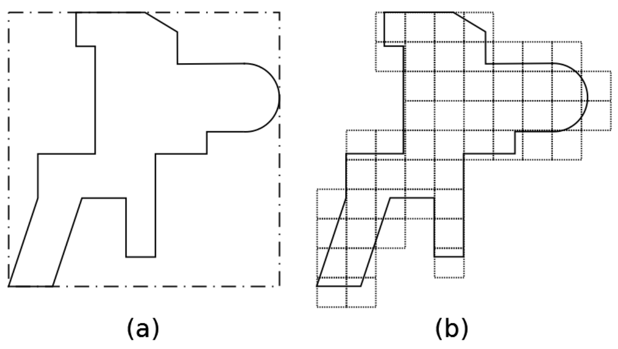

Figure 3.7: (a) Game world with bounding rectangle frame, (b) Game world with cell placement based on bounding rectangle frame

Table 3.2: Simulation Parameters

| Parameters | Value |
|---|---|
| $L * H$ | $4000 * 4000; 8000 * 8000; 12000 * 12000$ |
| $P$ | 128; 256; 512 |
| $v$ | 1; 5; 9 |
| $A$ | 100000; 150000; 200000; 250000; 300000; 350000; 400000 |
| $AOIRadius$ | 75 |

## 3.3 Simulation Result & Discussion

We simulated 4 common partition shapes (rectangular, hexagon, triangle and variable rectangle) and our BIP with a combination of parameters. We generated 10 random scenarios for each value of $L$, $P$, $v$ and $A$. The combination of simulation parameters can be found in Table 3.2. At the end of simulation, final result is obtained through the average value of $\eta$, $\theta$, $\nu$, and $\mu$ from all scenarios. The whole simulation processes are described in the Figure 3.8.

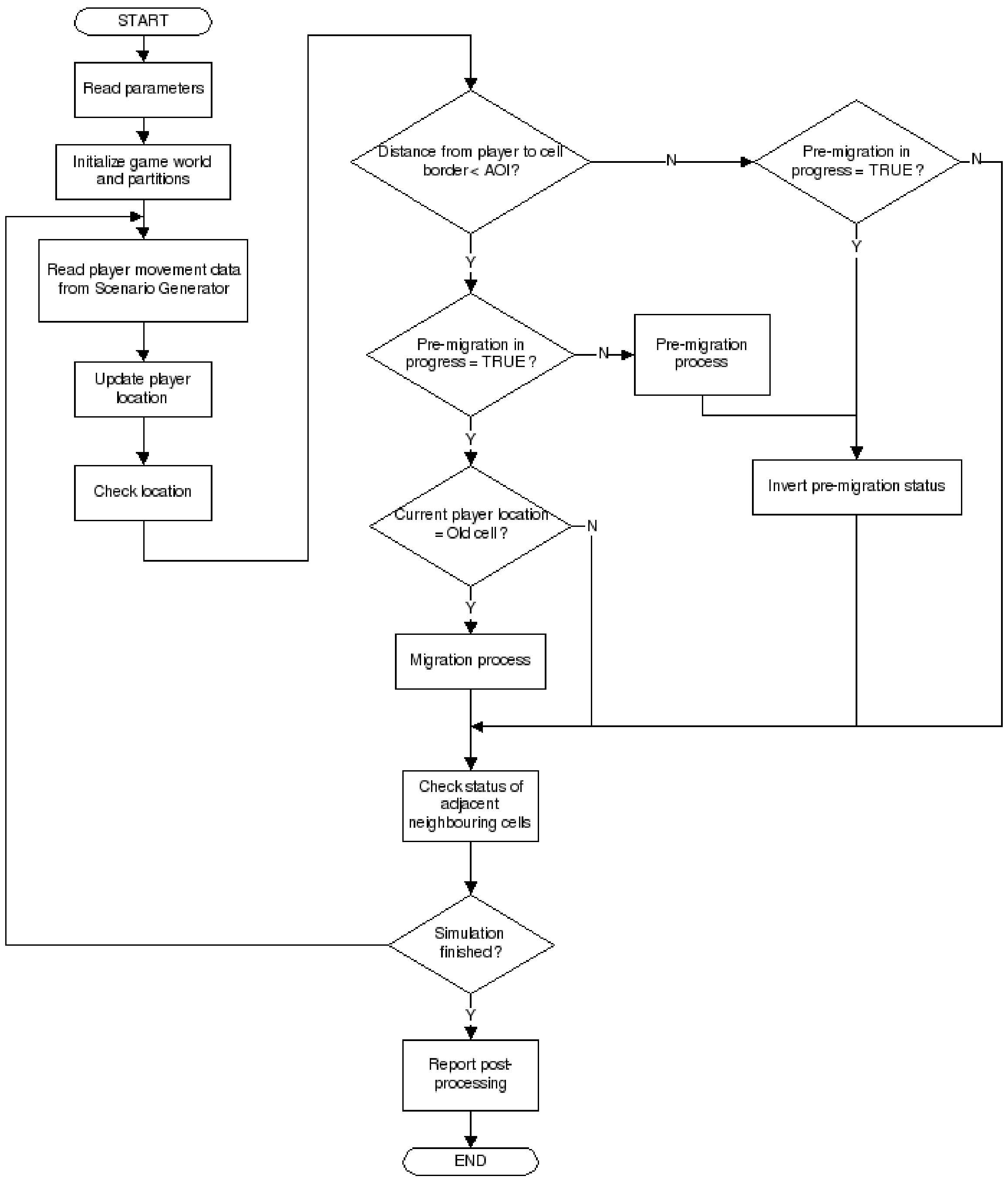
START
Read parameters
Initialize game world and partitions
Read player movement data from Scenario Generator
Update player location
Check location
Distance from player to cell border < AOI?
N
Y
Pre-migration in progress = TRUE ?
N
Y
Pre-migration in progress = TRUE ?
N
Y
Pre-migration process
Invert pre-migration status
Current player location = Old cell ?
N
Y
Migration process
Check status of adjacent neighbouring cells
Simulation finished?
Y
Report post-processing
END

Figure 3.8: GWP Game Implementation Diagram

Table 3.3: Minimum and maximum number of player in one cell for BIP during game session

| GW Size($L * H$) | $P(Min)$ | $P(Max)$ |
|---|---|---|
| 4000*4000 | 0 | 15 |
| 8000*8000 | 0 | 13 |
| 12000*12000 | 0 | 8 |
| 16000*16000 | 0 | 3 |

The performance evaluation for each partition method is based on the value of $\eta$, $\theta$, $\nu$, and $\mu$. We observed the performance and determine the most significant factor that could affect it. One thing to note is that better performance will result in more efficient network resource usage and in this case, lower value means better performance. During early simulations, we found that the size of game world does not affect the overall performance of all partition methods but instead reduces the player density of each cell where the impact can be seen on the number of minimum and maximum player in one cell (table 3.3). In the next part, we will present the analysis for other factors such as $P$, $v$ and $A$ and their impact on the value of $\eta$, $\theta$, $\nu$, and $\mu$.

## 3.3.1 Number of Player

Graphs in Figure 3.9 and 3.10 show how the number of players affects $\theta$ and $\nu$. Simulation observed in those figures use fixed value of $L * H = 8000 * 8000$, $v = 5$ and $A = 200000$. The number of players has linear correlation to overall performance. In every partition method, value of $\eta$ and $\nu$ are nearly doubled following the increment of number of player. This is to be expected even in real games where having more player involved in a game

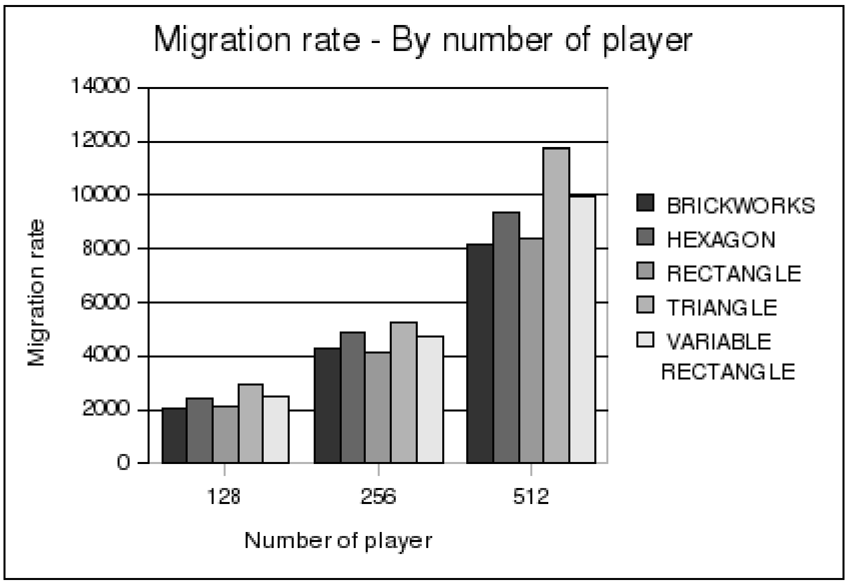

Figure 3.9: Migration rate with different number of player

session will increase possibility of having more events occurred. Next, for each of partition methods, triangle holds the lowest performance with highest value of $\eta$ and $\nu$. The variable rectangle follows the triangle partition for migration rate. However, despite of being high in migration rate, it shows that pre-migration rate is quite low compared to the others (10% lower compared to the hexagon). The result shows that different partition method has different pre-migration and migration rate ratio of $\eta$ and $\nu$. Meanwhile, hexagon partition method has lower migration rate compared to others, but it is not the same with pre-migration rate. BIP in this aspect behave similarly with other partition and the rectangle partition method has the lowest $\theta$ and $\nu$. The first evaluation supports our hypothesis that with different partition shape, it is possible to find the best one to increase efficiency in network resource usage. Overall result shows the pattern of $\eta$ and $\nu$ follows the value of number of player but not with $\theta$ and $\mu$.

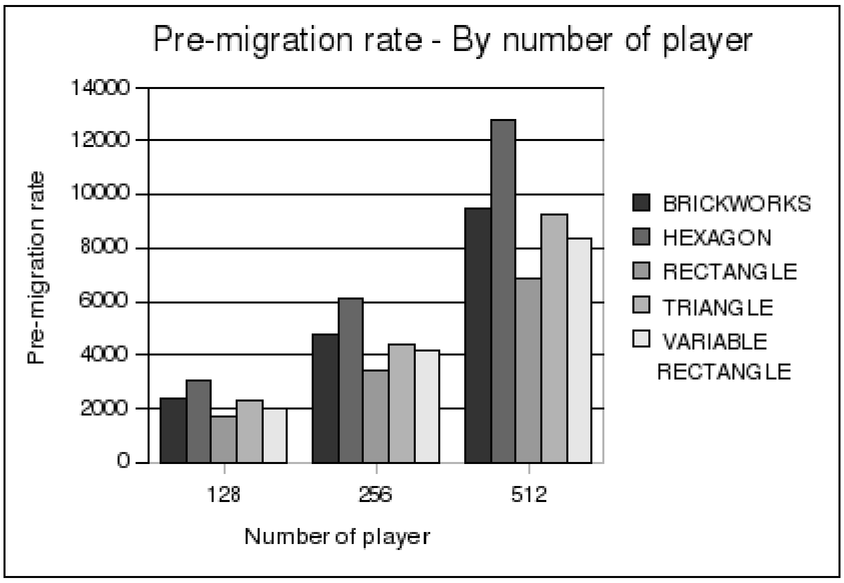

Figure 3.10: Pre-migration rate with different number of player

### 3.3.2 Player Movement Speed

Figure 3.11 and 3.12 shows that we can see a different pattern compared to previous result with number of players with variation in player movement speed, . While number of player goes linearly with the value of $\eta$ and $\nu$, this is not the case with player movement speed. As the speed increase linearly, the result jumps exponentially when we use maximum speed (9). Graph in Figure 14 shows the result for different value of $\upsilon$ (1, 5 and 9). We can see that the result does not remain steady like the one in number of player evaluation. However, the pattern of performance for each partition method remains the same. The evaluation shows that BIP always performs decently for the migration rate and lags behind for pre-migration. Again, even with speed, there is no similar pattern for $\theta$ and $\mu$.

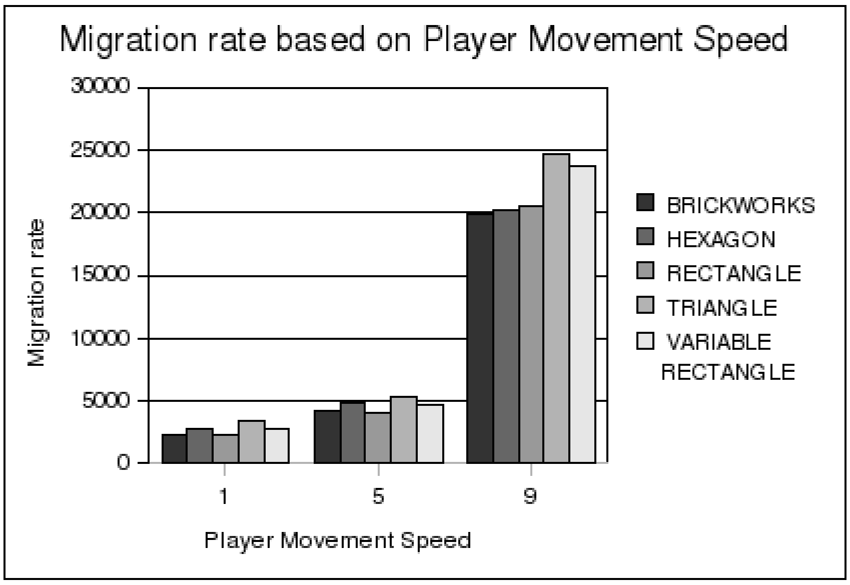

Figure 3.11: Migration rate with different player movement speed

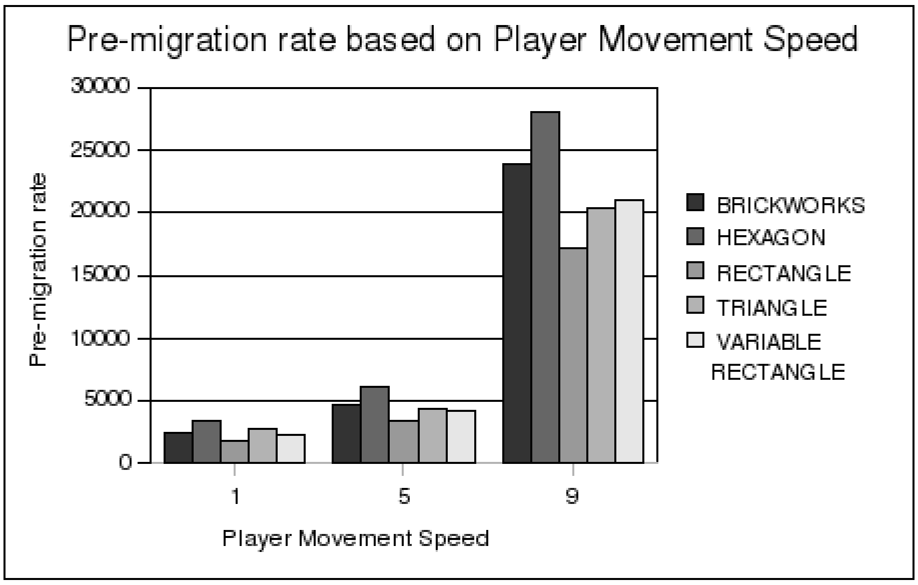

Figure 3.12: Pre-migration rate with different player movement speed

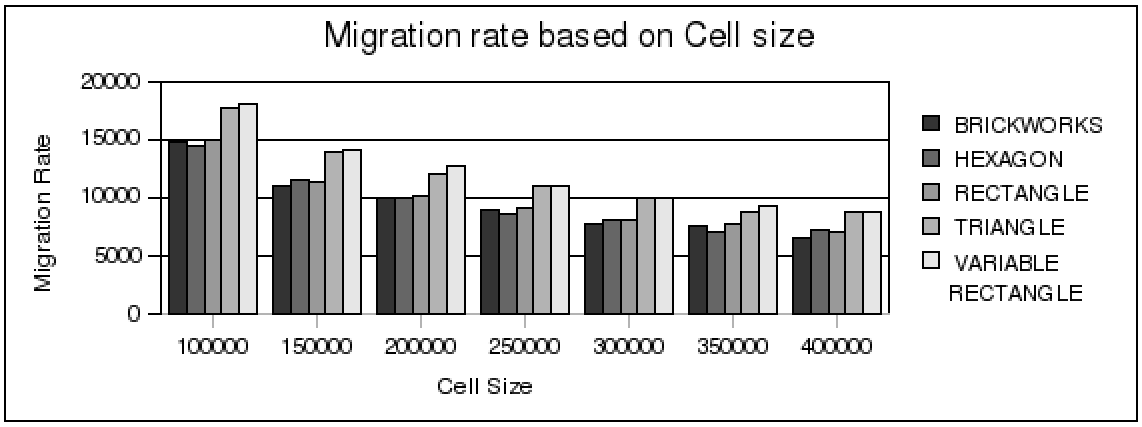

Figure 3.13: Migration rate based on cell size

### 3.3.3 Cell Size

Graphs showed in Figure 3.13 and 3.14 shows variations of cell size and the impact to $\eta$ and $\nu$. Overall result indicates improvement in lower value of $\eta$ and $\nu$ as the cell size increases. However, as the cell size gets even larger, there is a tendency that it will reach saturation point where it will no longer get a significant improvement compared to the memory resource usage. In this evaluation, the result shows that the $\nu$ of BIP performs similarly with hexagon partition method as the cell size increases. However, we can see significant improvement in $\eta$ as it starts with similar performance with hexagon but in the end, its performance rate is getting close to the other partition method. One thing to note is that rectangle partition performs better than others in this category.

### 3.3.4 Consecutive Pre-migration and Migration Rate

During simulation, we found no specific pattern of $\theta$ and $\mu$ against other parameters. However, the value of $\theta$ and $\mu$ were always between 10-20% in our 10 different random scenarios. For example, one of the simulations reported the final value of $\theta = 2045$ and $\mu$

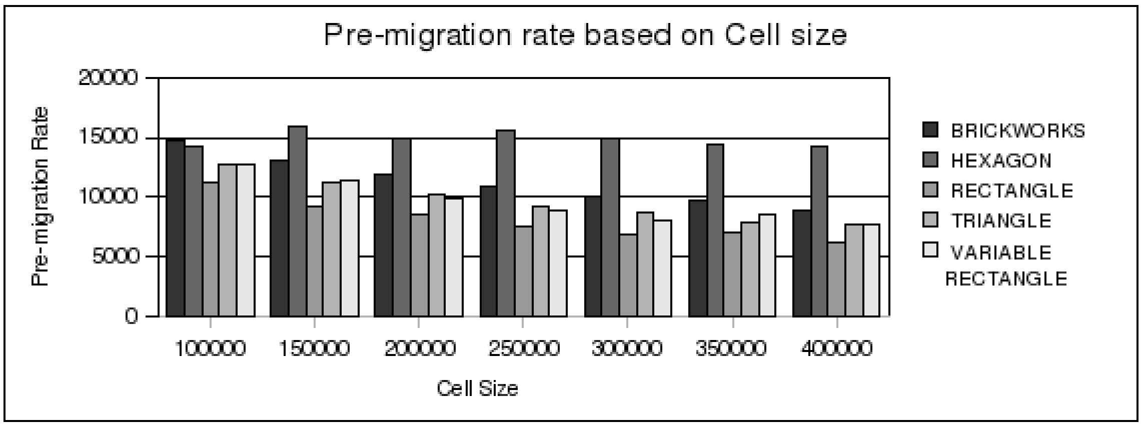

Figure 3.14: Pre-migration rate based on cell size

= 331; but in the other scenario with the same parameters, it recorded $\theta$ = 1731 and $\mu$ = 619. However, we are confident that based on our partition mechanism, our partition method still get the benefit during consecutive pre-migration events.

Based on our simulation results ($P$=256, $v$=5), the average number of other players in neighbouring cell during cell migration in BIP is between 7-10 players. However, this number is significantly reduced between 0-3 within the internal partitions crossed by the player's AOI. The general rule is: during CCPM events, we can expect increase in overall network efficiency compared to other methods especially with higher number of $\theta$ and $\mu$.

## 3.4 Summary

Our experiment result demonstrates various correlations of parameters for different game world partition shapes. However, as different gaming applications have different implementations or approaches, we present our result and conclusion as design consideration for implementation of multiplayer online games.

There are three main factors that determine the performance of game world partition methods. They are: player movement speed, cell size and cell shape. Our proposed solutions have similar performance with others in the simulation with high speed of player movements. However, our simulation shows that it requires lower number of packets transferred during game session because of its internal partition. It means we have better scalability and efficiency by having lower network resource usage while keeping game mechanics simple. Rectangle partition has higher performance in average, but it has higher possible neighbouring cells during the cell pre-migration and consecutive pre-migration processes which will have impact on the final network resource usage. Another finding is that game world size has no true impact to the performance. Nevertheless, it is still important to consider about the cell size related to the size of game world given that larger cell size will result in more requirements in memory to store cell state information while reducing the number of cells required covering the game world.

# Chapter 4

# Network Workload Evaluation in Game World Partition System

## 4.1 Introduction

The popularity of Massively Multiplayer Online Games(MMOGs) has grown rapidly over the last few years with some of the popular games like *World of Warcraft*, *Lineage 2* and *Ragnarok* having the trend of increasing number of active players. Advances in Internet network technology enable more players to play in one game by providing larger bandwidth and better management of network resources. The increasing number of players creates challenges in network resource usage evaluation for both game developer and game service provider [6]. Game developers always try to create more efficient network communication protocols and techniques in order to have better utilization of network resources. In this case, game developers need to evaluate new communication protocols or techniques before implementing them into games. Meanwhile, game service providers with game servers have to provide adequate network resources to support their MMOGs. In most situations, they always prefer overestimation rather than underestimation because underestimation could cause network lag and render the MMOG unplayable. However, game service providers also want to avoid high overestimation in order to minimise unused network resource and operational costs.

In this Chapter, we propose a mechanism to evaluate network workload for MMOGs. The main objective here is to design a flexible mechanism that could be easily modified and expanded to accommodate different implementation of MMOGs. Our approach to the solution is by using combination of mathematical model and simulation. We will use a specific game system setup and mechanic to demonstrate our proposed solution. Our game system assumed in this chapter uses Game World Partition (GWP), an Interest Management System (IMS) that divides game world into sections with uniform shape. For the sake of simplicity, in this Chapter, we refer GWP as IMS.

## 4.2 Game System Description

Our description of multiplayer online game system is based on the general assumption of common MMOGs where the system consists of a number of entities. These entities have different types based on their role and network communication architecture. We consider the system as one big collection of state machines where each of those state machines belongs to one single entity. One type of entity that always exists in every MMOG is the player. As mentioned in previous chapter, the term *player* refers to the machine used by human player (user) to play the game. During gameplay, players will need to send and receive game state updates in order to keep their game state up-to-date with other

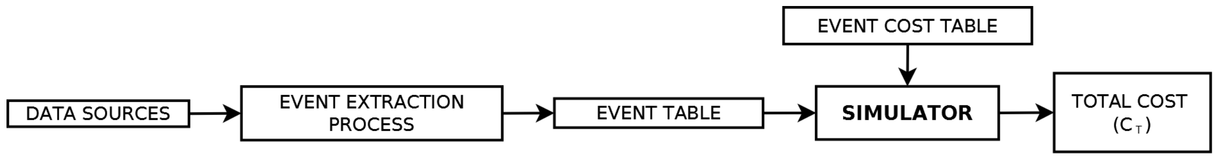

Figure 4.1: The mechanism for network workload evaluation

entities. These activities are reflected in network workload as sending and receiving game state updates require the consumption of network resources such as bandwidth. There are different ways to represent the network workload such as using the number of network packets transferred during the game, the number of bytes transferred or workload index defined by the game developer. We are using the number of packets transferred, defined as $C_T$, to represent the total network workload for the whole game session.

To obtain $C_T$, our mechanism uses the combination of mathematical model and simulation. The general process of our mechanism is described in Figure 4.1. We assume a universal time instance for all entities in the game system. The time instance is represented by $t$ ($t = 1, 2, ..., T$) where $T$ is the finishing time of a game session. The unit measurement of one time instance is to be defined in advance. We will explain our assumption of the time instance used in this chapter in the next sub-section.

The first part of our mechanism is the data source. Data source is basically all events that will trigger the use of network resources such as sending or receiving game state updates. There are several options for the data source such as: random data, pre-recorded game session and data generated from other mathematical model, e.g. Hidden Markov Model (HMM). Game service providers can use previously recorded game session to estimate the network workload in more accurate manner. At the same time, game developer can use the same data source to test the efficiency of their new communication protocols

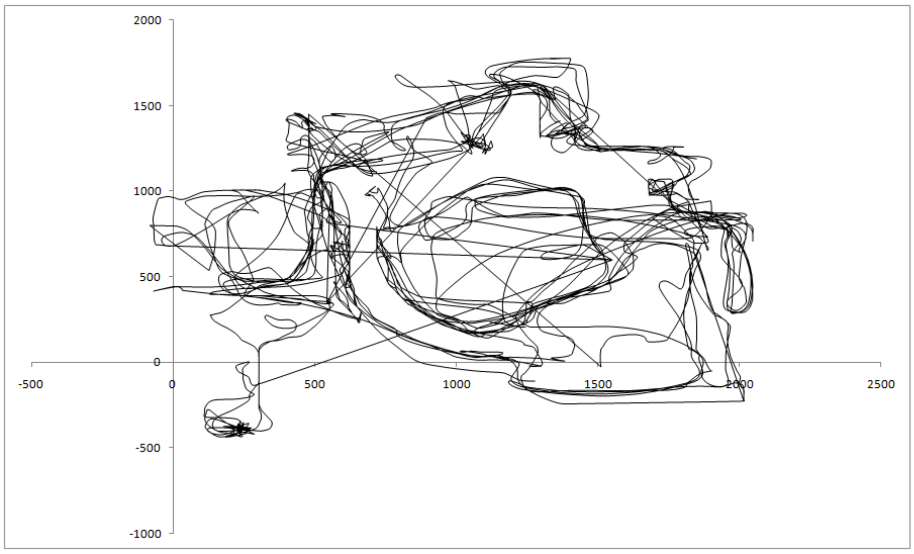

Figure 4.2: Movement traces (1)

or IMS. Additionally, our mechanism can be implemented by other researchers developing network traffic model to validate their work by comparing it against the result from other data sources.

In our simulation, we use real data, which is the recorded game sessions of Quake 2 from [72]. We extract the movement traces of different players during the game and use them as our data source. Figure 4.2, 4.3, 4.4, and 4.5 depict movement traces of 4 different players in the same map called 'The Edge '. At a glance, it is clear that each player has different favorite places to visit and stay. There are also places which generally attractive for all players and where the main duels between players happen frequently. Once we can identify these places, we can implement better network provisioning system in order to cope with higher network resource requirements during online game session. However, there is one limitation of our data source: it does not record the actions of other players in detail. Therefore, we use different game recording file with the same map to simulate multiple players in our research.

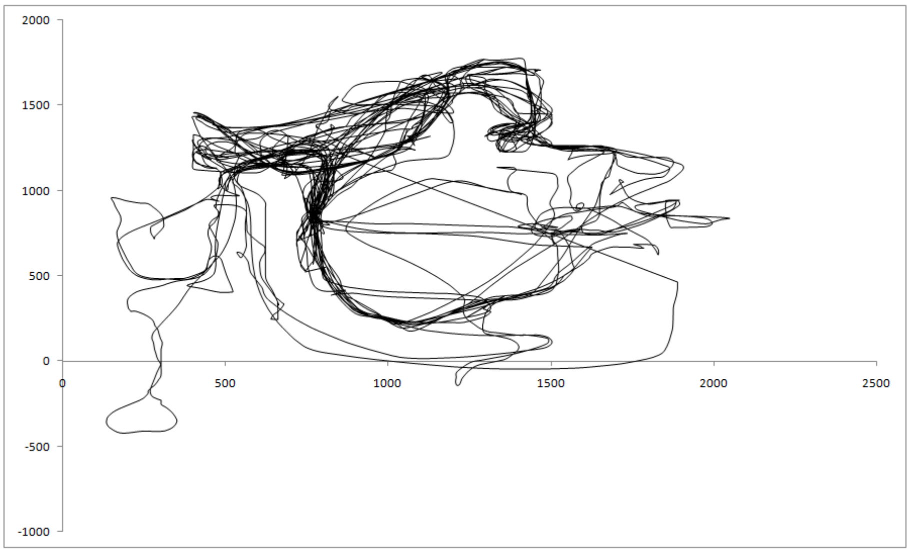

Figure 4.3: Movement traces (2)

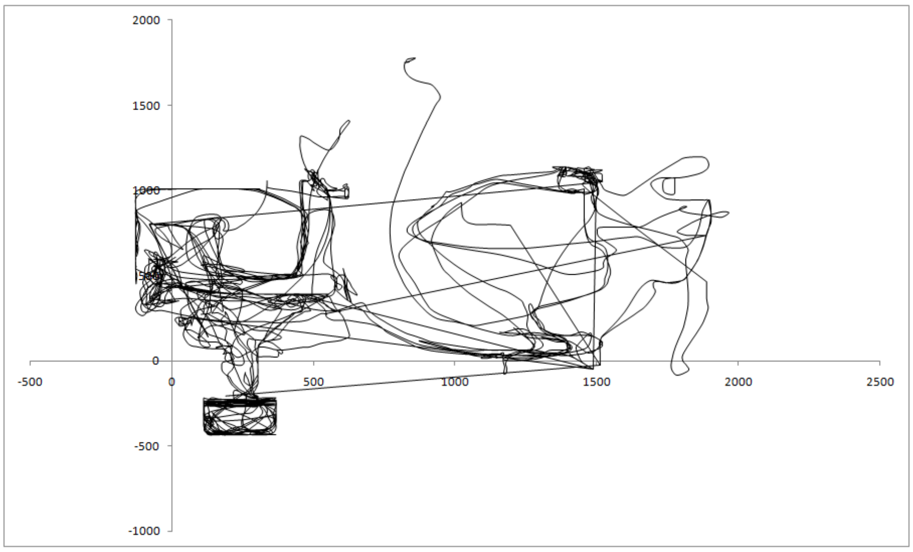

Figure 4.4: Movement traces (3)

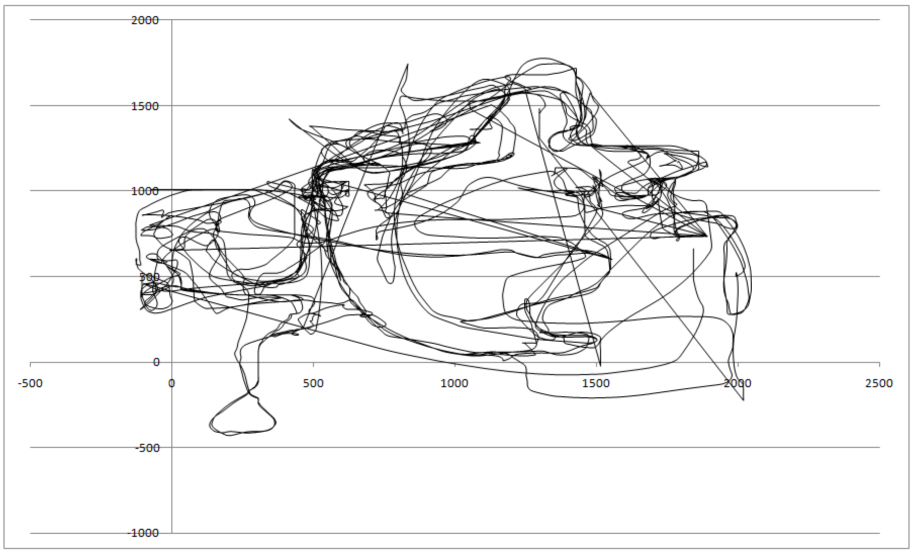

Figure 4.5: Movement traces (4)

The second part of our mechanism is Event Extraction Process. At this stage, the data input is extracted and converted into a series of events according to the game mechanic. The second part is the crucial part of our mechanism. The reason is because we have to define the type of events that we want to extract from data source and the cost-function related to each of the event type. All of the extracted events will be stored in a table called Event Table (*ET*) as described in Table 4.1. The upper boundary value for Event Type field is $k$, which is the maximum number of event type. The field Event Type in *ET* acts as the table key that connects to another table, Event-Cost Table (*ECT*) as described in Table 4.2. *ECT* contains two fields, the first one is Event Type and the second one is the network cost function. The cost function can be defined in various forms, ranging from a constant, linear functions or even more complex functions. This part can be viewed as the translation process where it will extract events from the data source and stored into the tables.

The third part is the simulator that will read data from *ET* and *ECT*, then put the

| **Field name** | **Value** |
|---|---|
| Record Index ($g$) | $\{1, 2, ..., I\}$ |
| Time Instance Index ($t$) | $\{1, 2, ..., T\}$ |
| Event Type($l$) | $\{1, 2, ..., k\}$ |

Table 4.1: Structure for Event Table

| **Field name** | **Value** |
|---|---|
| Event Type ($l$) | $\{1, 2, ..., k\}$ |
| Network Workload($f_c(l)$) | $\{1, 2, 3, ...\}$ |

Table 4.2: Structure for Event-Cost Table

entry into mathematical formula to calculate the final result ($C_T$). We will explain more detail about this in mathematical model section. Next, we will elaborate more about our game system. What we are using in this research is just an example of how to use our mechanism. Other type of games will have different event type and cost-functions. In our simulator, we use objects to represent entities and to store data. The attributes for entity object can be seen in Table 4.3. With every entity storing their own events, we can calculate the network workload globally and individually.

These three main parts of our mechanism are the ones, which can be modified if we want to implement a game system into our mechanism. We will explain our implementation of game system in the next sub-sections.

| **Attribute** | **Type** |
|---|---|
| ID | integer |
| ET | EventTable (Table 4.1) |

Table 4.3: Entity Object Attributes

### 4.2.1 Communication Architecture

In our simulation, we use Client-server (CS) communication architecture which means there is another type of entity: Game server. As explained in Chapter 2, the game server and players have their own loop which consists of several processes. For the game server, the loop starts at the same time as the beginning of a game session. For the player, the loop starts when the player join an online game session. We use the assumption that the *tick* of game server happens at the same time as the player.

In our simulation, we do not consider network lag/delay and packet lost but instead we are using the assumption of an ideal network. Once a source entity send a packet at a specific time $t$, the destined entity will receive it at the same time. However, our $ET$ can accommodate delay if required. It can be done by modifying the $t$ of the $ET$ entry for a specific event. For example, player A sends a packet to player B at tick 52. To simulate a network delay of 4 ticks, the system will adjust the $t$ value for player B for the receive event to 56. It shows that player B does not receive the packet at the same time it is send by player A.

### 4.2.2 State Update Mechanism

In order to keep game state synchronized, all players in the MMOG have to communicate their state update to game server and in return, receiving other players' state update. We assume that a player always send the state update every one tick. This is quite similar to the methods used by modern multiplayer game engines such as *Source Game Engine* [80]. Source Game Engine uses the default value of 66 ticks per second. Based on our data source, we are using 10 ticks per second as the rate for the game state update. It means that players will send and receive state updates from game server every 100 millisecond. On top of this, there are special state updates for IMS events that we will explain later in this chapter.

### 4.2.3 Game World Partition System

As described in Chapter 3, Game World Partition mechanism used in our research is the one that divides game world into different sections that we call cells. Each cell is shaped uniformly with the other. These cells will be placed following tiled pattern. The purpose of GWP is to provide better network resource management by creating a filter based on spatial location of players. Players in different cells are not required to communicate to each other, only players in the same cell need to synchronize their state updates. In practice, communicating state updates without IMS in MMOGs is very costly for game servers and players.

Figure 4.6 shows an example of 4 cells with 4 players with square cells. In CS network architecture, all state updates from players will be send to game server. After updating global game state, the game server will send specific state updates to all players according

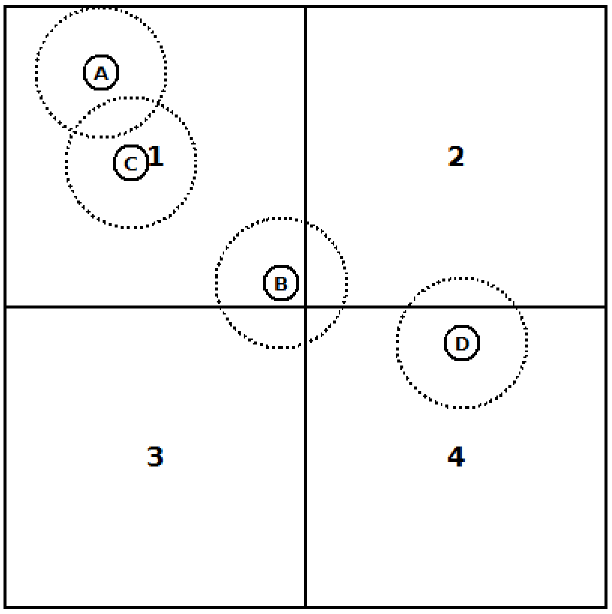

Figure 4.6: Game World Partition with 4 Players and 4 Cells

to IMS used in the game. In Figure 4.6, player A, B and C are located in cell 1, therefore player D will not receive state updates from player A, B and C because player D is located in cell 4. Without IMS, the number of packet transferred in one state update is $n^2$, where $n$ is the number of players and $n > 1$. By using GWP, the number of packet transferred in one global state update is variable which depends on the position of players in the game world. In Figure 4.6, the cost of one global state update is $3^2 + 2 = 11$ packets. We will explain the detail for the formula in the later section.

### 4.2.4 Events of Game World Partition System

We have mentioned in the previous section that GWP works by dividing game world into smaller sections that we call cells. It is common for a player to move from one cell to the other during a game session. The responsibility of a game server is to keep track of the location of every player in the game world. Game server also determines which part of state update need to be sent to which player. For the players, all they need to do is to send their state update and to receive state updates of their cell from the game server. Based on our game mechanics, there are two main events that invoke network activity: regular state update and special event state update. Regular state update requires player to send and receive state updates from game server *periodically.* Special event state update is defined by other events in game mechanics that actually trigger network activities.

As shown in Figure 4.6, there is a circle surrounding a player. This circle is what we call *Area of Interest*(AOI). As the player moves during game session, the edge of AOI might touch the boundary of another cell. When this happens, game server will start to send partial information of the other cell to player $A$. This event is what we define as *pre-migration event*, which we simply call PME for the rest of the chapter. PME may be cancelled if the player moves further from the cell boundary and going back inside the home cell. However, it is also possible that a player moves to the other cell after pre-migration event occurred. This event is what we call *migration event* (ME). During ME, the player will send a special state update to the game server. Then the game server will update its global game state and the cell membership list. Next, the game server will send the remaining information of the new cell to the migrating player while at the same time sending the state update to all players in the old and new cell that there is a player changing its home cell. $E_{PM}$ and $E_M$ are the special events in our mechanism and we will define the formula to calculate their costs in the next section.

## 4.3 Mathematical Model

This section provides the details about our mathematical formula, including the components inside it. Our aim is to create a flexible model to cope with different type of games. To explain our model, we define the following notations:

- $n$: Number of player in current game session.
- $N$: Number of cell in a game world.
- $t$: Time indicator in ticks ($1 \leq t \leq T$).
- $j$: Cell index indicator ($1 \leq j \leq N$).
- $a$: Indicates the current cell index during PME which is also the old cell index during ME.
- $b$: Indicates the destination cell index during PME which is also the new cell index during ME.
- $i$: Player index indicator ($1 \leq i \leq n$).
- $\theta$: The network workload to send or receive one state update. In this thesis, we simply define $\theta = 1$ for our simulation.
- $f_{cc}(j, t)$: Function that returns the number of player in cell $j$ at time $t$.
- $\mu(j, t)$: Function that returns the network workload of sending and receiving one state update of cell $j$ at time $t$.
- $\alpha$: The ratio of transferred cell information during PME ($0 \leq \alpha \leq 1$). In our simulation, we define $\alpha = 0.5$.

- $\sigma(i, t)$: The function that returns the network workload of a PME of player $i$ at time $t$.
- $\tau(i, t)$: The function that returns the network workload of a migration event of player $i$ at time $t$.
- $E^i$: The matrix that stores all events of player $i$ extracted from data source. We define the matrix as:

  $$E^i = [e_{kl}]_{R_i \times 2}$$

  where $R_i$ is the total number of events generated by player $i$. The columns of matrix $E^i$ corresponds to the structure of Table 4.1 which means the first column($e_{k1}$) stores the time when the event occurred and the second column($e_{k2}$) stores event types. Table 4.4 shows the constant values for event types.
- $C_S$: Network workload for the game server throughout one game session.
- $C_P$: Network workload for all players throughout one game session.
- $f^s_{C_s}(t)$: Total network workload of a game server to send state updates at time $t$.
- $f^{sp}_{C_s}(i, t)$: The network workload of a game server to send a state update to player $i$ at time $t$.
- $f^s_{C_r}(t)$: Total network workload of a game server to receive state updates at time $t$.
- $f^p_{C_s}(i, t)$: Total network workload of player $i$ to send state updates at time $t$.
- $f^p_{C_r}(i, t)$: Total network workload of a player $i$ to receive state updates at time $t$.

Based on the general process in multiplayer online games using CS architecture, our assumptions in this chapter are:

| $e_{k2}$ | Event type |
|---|---|
| 1 | Regular state update |
| 2 | Pre-migration |
| 3 | Migration |

Table 4.4: Constant value for event types

- In every tick, there is a regular state update from one player to the game server. This applies to all players in the game.
- In every tick, all players will receive relevant state updates from the server. The size of state update received from the game server depends on the implemented IMS.
- Pre-migration and migration events are not included in the regular state update. Therefore, there is an additional network workload whenever those events occur.

To get the total network workload for the whole game session, denoted by $C_T$, we need to add the total network workload of the game server and all players. Hence, we define $(C_T)$ as:

$$C_T = C_S + C_P \tag{4.1}$$

Next, we will derive the rest of the formulas for computing $C_S$ and $C_P$. Each of them includes the total of network workload to send and receive state updates during a game session. To get the total network workload for the whole game session, we need to calculate the network workload in every game tick from the beginning until the end of a game session.

Therefore, we can expand equation 4.1 into:

$$C_T = \sum_{t=1}^{T}[f^s_{C_s}(t) + f^s_{C_r}(t) + \sum_{i=1}^{n}[f^p_{C_s}(i,t) + f^p_{C_r}(i,t)]] \tag{4.2}$$

Equation 4.2 is generally applicable to the common MMOGs. The next step is to design the formula to accommodate different game systems. This includes the modeling of IMS, state update mechanisms and event cost definitions.

### 4.3.1 Network Workload Evaluation for Game Server

In our game system, the game server will receive state updates every tick from all players in the game. For comparison, we will include the formula to calculate network workload in a game system without IMS. First, we can define the network workload for the game server to receive the state update at time $t$ as:

$$f^s_{C_r}(t) = n\theta \tag{4.3}$$

After updating global game state, the game server need to send specific state updates to the relevant players. It means that every player will have to receive the state update of other players. Thus, the network workload to send the state updates to all players in one tick can be defined as:

$$f^s_{C_s}(t) = n(n-1)\theta \tag{4.4}$$

By adding Equation 4.3 and 4.4, the total network workload every tick for game server in a game system without using IMS is $n^2\theta$. $n$ is a variable which is initialized at the beginning of each game session and it will remain constant during the game session.

Meanwhile, in a game system using GWP, we need to calculate the network workload in a different way. Instead of sending and receiving the state update blindly, the game server needs to send specific game state updates to specific players. $f^s_{C_r}(t) = n\theta$ still applies in GWP system, because the game server need to receive state update from all players every game tick. However, we need to modify $f^s_{C_s}(t)$ into:

$$f^s_{C_s}(t) = \sum_{i=1}^{n} f^{sp}_{C_s}(i,t) \tag{4.5}$$

The formula above shows that the network workload for the game server to send state updates is different for each player. Since GWP divides game world into different cells, we define the network workload of state update for cell $j$ during time $t$ as:

$$\mu(j,t) = \begin{cases} f_{cc}(j,t)^2\theta & \text{if } f_{cc}(j,t) > 1, \\ \theta & \text{if } f_{cc}(j,t) = 1. \end{cases} \tag{4.6}$$

In Formula 4.6, the network workload in one cell is calculated in the same way as in the game system without IMS with two conditions. The first condition is where there is only one player in cell $j$ and the second condition is where there is more than one player in cell $j$. The improvement in efficiency is gained from the fact that $f_{cc}(j,t) < n$ in almost all cases. For example, in a game session where $n = 10$ and $N = 2$, the total network workload of the game system without IMS would be $n^2\theta = 100\theta$. The only worst case where the total network workload in the game system with IMS would be equal with the game system without IMS is when all players gather in one cell. Otherwise, the total network workload in the game system with IMS will always lower than the game system without IMS.

As explained in previous section, we have three different types of state update: regular state update, state update during PME and state update during ME. We use matrix $E^i$ to store all of those events for all players which in the end determine the network workload required for every player. For each row in $E^i$, we can define $f^{sp}_{C_s}(i,t)$ as:

$$f^{sp}_{C_s}(i,t) = \begin{cases} \mu(j,t) & \text{if } e_{k2} = 1, \\ \mu(j,t) + (\alpha\mu(b,t)) & \text{if } e_{k2} = 2, \\ \mu(a,t) & \text{if } e_{k2} = 3, \end{cases} \tag{4.7}$$

Formula 4.7 refers to the value of $E^i$, where $E^i \in e_{k2}$ and the values of $e_{k2}$ are defined in Table 4.4.

## 4.3.2 Network Workload Evaluation for Players

Similar with the game server, all players in the game will send and receive state updates every game tick. As defined before, all players will need to send their state update regularly. Therefore, we always have $f^{p}_{C_s}(i,t) = \theta$ during the gameplay for game system using IMS or without IMS.

To calculate the network workload to receive state update for a player $i$ in a game system without IMS, we can define it as:

$$f^{p}_{C_r}(i,t) = (n-1)\theta \tag{4.8}$$

Meanwhile, just like with the game server, we need to differentiate the formula to calculate network workload to receive state update during different events. Because we use the same $\theta$ value for sending and receiving state update in our game system, we can use the following formula:

$$f^{p}_{C_r}(i,t) = f^{sp}_{C_s}(i,t) \tag{4.9}$$

Substituting formulas (3)-(9) into formula (4.1), we obtain:

$$C_T = \begin{cases} 2n^2T\theta, & \text{without IMS,} \\ 2[nT\theta + \sum_{t=1}^{T}\sum_{i=1}^{n}[f^{sp}_{C_s}(i,t)]], & \text{with IMS.} \end{cases} \tag{4.10}$$

Formula 4.10 is the final result to calculate $C_T$ in two game systems, without and with IMS. As a reminder, the term IMS refers to GWP and other methods of IMS would have different formulas and variables.

## 4.4 Result & Analysis

In this section, we divide our result and analysis based on different change in parameters. The aim is to observe the impact of those changes in parameters against the growth of total network workload for the whole system.

Figure 4.7 shows the result of two different scenarios. Due to the length of game session, we present the cost from the beginning of game session until game tick 600. With different scenarios, there is small variation of the network workload throughout the game. Both scenarios have the same number of player and playing the same map. We can see that sometimes network workload will jump up by several points. This is due to the pre-migration event where a player prepares to move to another cell.

In Figure 4.8, we compare the network workload of game mechanics using GWP and without IMS by using the same scenario. With only 6 players, the network workload is

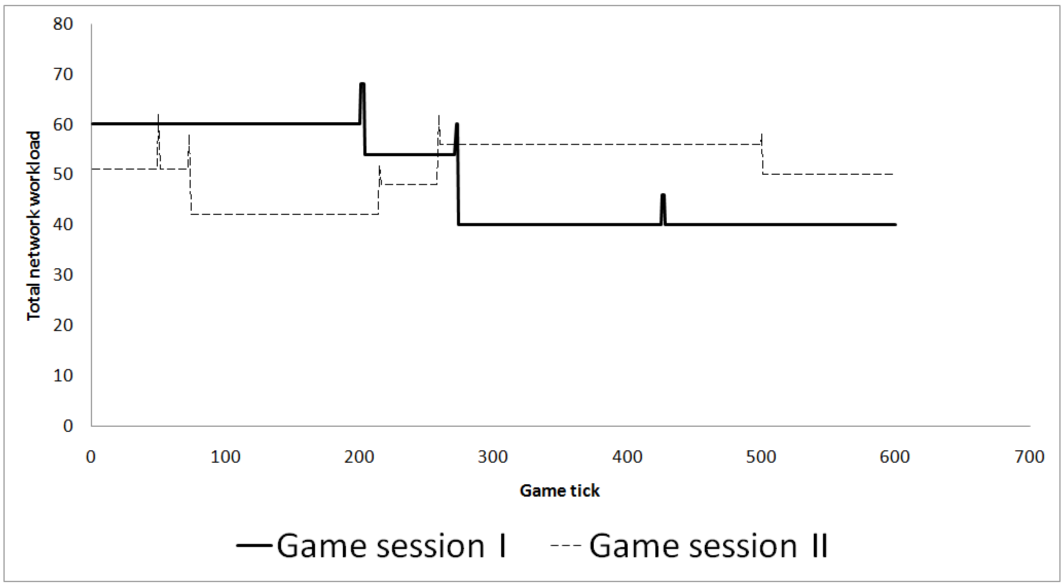

Figure 4.7: Network Workload for two different scenarios

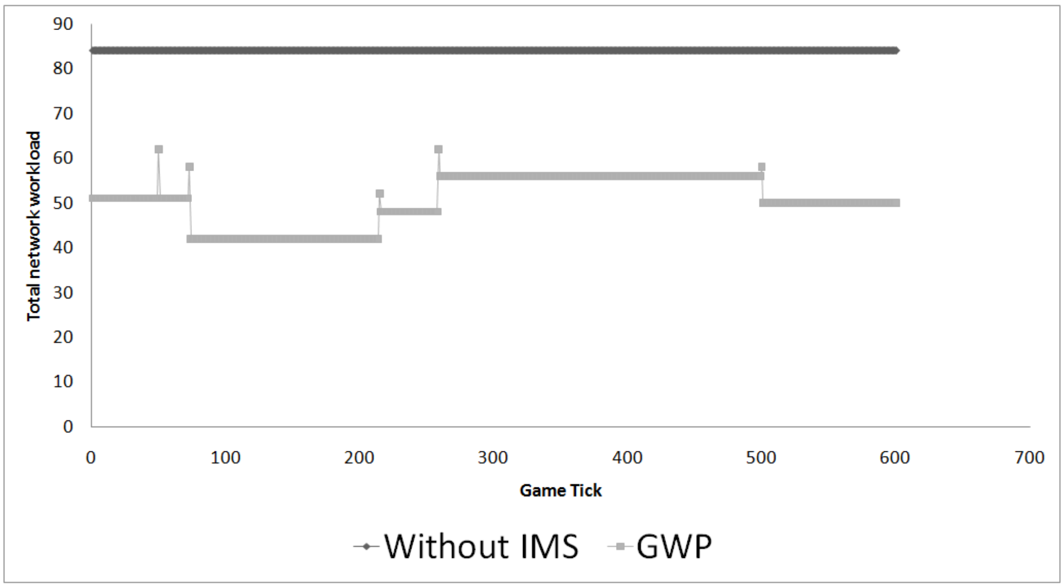

Figure 4.8: Network Workload with in scenarios with GWP and without IMS

almost doubled. For players in the game, network workload are much lower for the game using GWP compared to the one without IMS. Therefore, we can see that IMS helps to reduce the network cost as we have more players in the game. At the end, we have higher

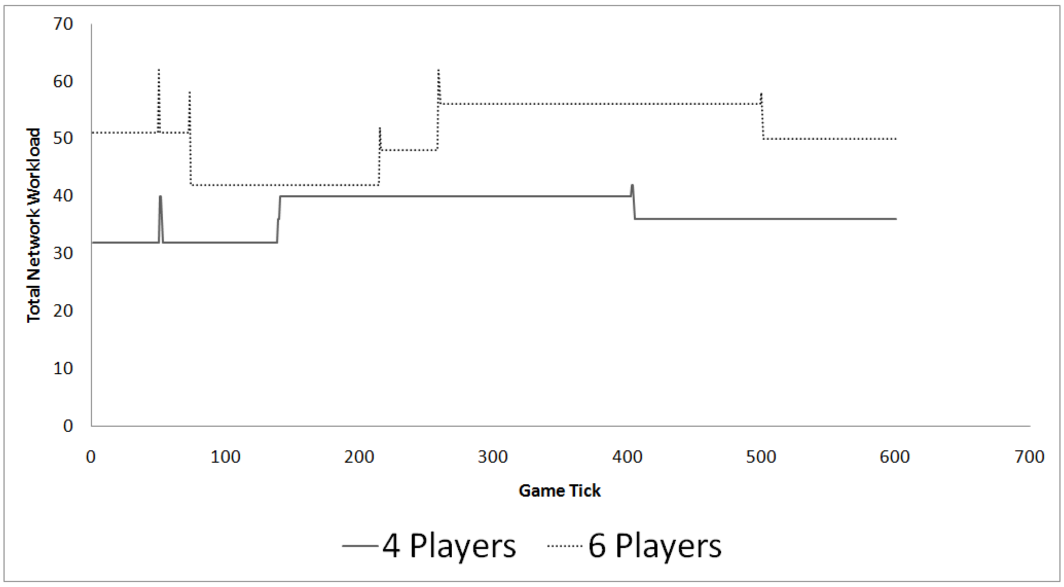

Figure 4.9: Network Cost for game scenario with 4 players and 6 players

game scalability and we can simulate the effect of adding more players to the total network workload. However, we also observed different patterns in our simulation as shown in Figure 4.9. There are some simulations that show the network workload of a game session that has more players is not much higher compared to the one with fewer players. We believe that this is due to the players' behaviors during the game.

One of the interesting features that we extracted from our formula is the ability to calculate the network workload down to the cell level. In Figure 4.10, we added several scenario of the same map all together so the number of player reaches 32. During simulation, we observed the network workload to get state update for each cell in the game world. Figure 4.10 shows network resource usage in cell 1, 5, 9 and 15. We can see the difference between cells and it is obvious that cell 1 has the lowest cost where during the game, there is only one player that does not move at all in cell 1 during the first part of the game. It translates to the conclusion that the game world covered by cell 1 is not a

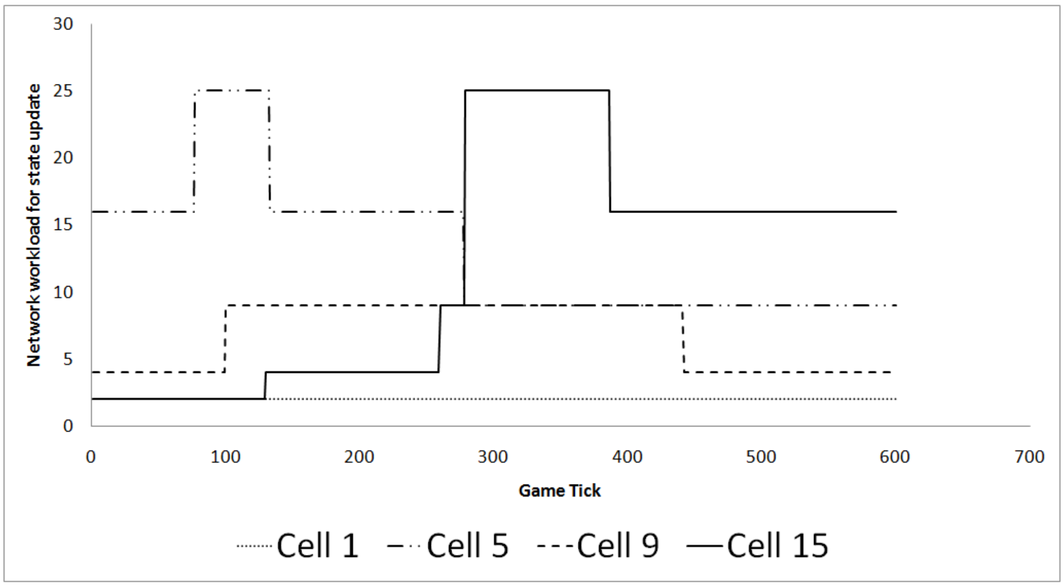

Figure 4.10: Network Cost for different cells in a game scenario

popular spot for players. On the contrary, cell 9 which is located almost in the middle of the map, has the highest cost. Further examination shows that it has the most number of players during simulation. Therefore it brings the conclusion that cell 9 is the 'hot-spot 'spot of that particular map. A hot-spot is a popular area in the game map, resulting in higher network resource usage for the players in the particular area. Using this analysis, we can point out the cell that needs extra network resources and which cells can be assigned to game server with lower network resources.

Another usage of our mechanism is AOI radius analysis in GWP. During our simulation, the radius of AOI is 150. However, we found that in average, ME occurs in 4 game ticks after PME. Since GWP relies on PME to provide better cell transition, migrating entities in the game may need more time to transfer the partial of cell state. We can fix the problem by having bigger AOI radius. As we increased our radius to 300, the average time between PME and ME becomes 8 game ticks. However, the number of ME also increases as the player has more possibility to change cells although the increase is not significant.

Figure 4.10 shows that we can analyze which cell is the most or least popular. This analysis opens to the opportunity for future research where we can have a variable size of cell instead of uniform size for GWP. The very same mechanism can be used to evaluate and compare the network workload.

By using event tables, we are able to use various event data sources from recorded game data, other mathematical/statistical formulas. We begin the work by identifying the basic concept of all multiplayer games, which is game event. No matter what the type of the games, they always have events and those events trigger the use of network resources.

An expansion of our framework is to use it for delay analysis. As explained in the previous section, our event table contains game tick. In our simulation, we do not include or consider network delay. However if necessary, delay can be added by adding the delay to the game tick. For example, when player 1 send a state update at game tick 15, the server receives it at game tick 17 ($\Delta t = 2$). By changing the entry to the event table of game server, we can simulate the similar effect of network delay. Furthermore, coupled with other research work to make model for network delays, we can get a better result instead of using random number to simulate delay.

Another expansion is the implementation of our framework to peer to peer or hybrid network architecture. We definitely need to modify the name of entities and game mechanics, but not the basic concept of the framework. To accommodate different network architecture, we just need to modify the procedure to fill the event table according to the data communication pattern of those network architectures. The same concept also applies to the multiplayer games that implement different game mechanics (e.g. different IMS). All we need to do is just to extract the events of those systems and put them into our event table. Additionally, we also need to modify the event cost table associated to the events.

## 4.5 Related Work

There are not many formalised mechanism to measure network workload in MMOGs. While there are many researchers presenting their proposed solutions, the method to analyze and to compare their work with others often differs between one another. For example, there are other researchers who developed mathematical model to measure the performance of their work. [74] designed a simple mathematical model to calculate network workload against the number of player in a P2P network. Other similar research [16, 25, 96, 97, 98] includes their own mathematical methods to measure the performance of their proposed solutions. Other researchers such as [17, 51] implemented a direct measurement method by setting up network listener between game server and players. The differences in their method to measure inspire us to design our mechanism that has the potential to be used as a standard to give researchers better view to the performance analysis of their work compared to others. Additionally, research that implements direct measurement of network workload can extract their data in a standardized way so other future research can use the same data in order to give better insight of the performance.

In relation to other research about network communication protocols [38, 44, 55], our mechanism can be used to measure their network workload. In the end, it is not only to compare the total network workload, but to identify other factors such as advantages and disadvantages of a communication protocols in a certain situation and the overall efficiency of a communication protocol in different type of games.

## 4.6 Summary

According to the simulation result, we can use our formula to see the impact of various parameters of multiplayer game elements to the total network resource cost. Although our target environment is specific to multiplayer games using game world partition system as their IMS, we believe that it can be modified easily to cope with other IMS. Our formulas have modular elements and IMS cost calculation is just one of the modules. Furthermore, all multiplayer online games follow the same principals that have been covered within our model.

Using our mechanism, we can pinpoint the network workload for different element of the game system. This could help game developer and other research to create and test new network algorithms and communication protocols. For game service provider, our mechanism can be implemented to identify network workload in various scenario and different type of games. The result may help game service provider to plan their network resources better in order to cope with the growth of number of players (improving scalability).

Finally, we want to emphasize that the main use of our simulator, techniques and formula in game development is to assist with resource planning and provisioning.

# Chapter 5

# Artificial Neural Network for Dead Reckoning in 3D Games

## 5.1 Introduction

One of the common problems in Multiplayer Online Games is network lag and disconnections between server and clients or between players. During network lag or disconnection, there is information loss or delay so the entities in the game cannot receive the information when they need it. Various methods attempting to solve the problem caused by network lag have been available and implemented in various multiplayer online games. In this chapter we are looking at the solution at the user-experience level by minimizing the effect of network lag to the gaming experience. Our approach to the problem above is by using Dead Reckoning (DR) in order to provide extrapolated values of player attributes whenever there is no data available due to network lag.

Unlike common DR methods, we are incorporating Artificial Neural Network (ANN) to get the extrapolated value. The design and implementation of ANN is further explained in this chapter in later section. Our research limitation in this chapter is defined by the type of the game and game mechanics. Because there are a huge variations in implementations, we have to limit the scope of our research into more specific type of games and their mechanics. Another reason for this is because the impact of network lag and disconnections is more severe in certain types of games and game mechanics. We are going to discuss this further in the later section along with the explanation for Dead Reckoning.

Because Dead Reckoning methods do the main calculations in players' hardware, it creates limitations in terms of available computing resources. Every game needs various resources in client's hardware to keep it running smoothly. Therefore, we cannot use DR method that consumes large resources. This constraint becomes one of the fundamental design challenges in our proposed solution. In order to test our design, we have implemented it into our simulator that also acts as the trainer for the ANN. As for the training data, we obtained all of those from recorded game data of *Quake II* [72]. Furthermore, we created formulas to weight the error rate of DR methods according to our research constraints in order to observe the efficiency of our methods. After the training, we obtained more detailed data of player movement by using modified *Jake2* [79] which is basically Quake II game that has been converted to Java. We use the captured data to compare the performance of standard DR method and our method.

## 5.2 Impact of Network Latency in Multiplayer Online Games

Multiplayer online games are relying on the network connection to exchange information between player-to-player or player-to-server. The exchange of game state updates is important to keep game data/state synchronized between entities in a multiplayer game. In this chapter, we use the term *entity* to represent any machines such as game server and computers used by players to play the multiplayer game. Some types of online games suffer more impact of network lag than others. This is true especially for real-time games where all players do the action simultaneously and they need to receive state update on-time. When network lag occurs, one entity cannot send or receive state update in a timely manner to and from other entities. Without up-to-date game state, a player cannot perceive game world accurately and it could lead to reduced game play experience. One real example is a multiplayer online game called *Armored Core: For Answer (ACFA)*. ACFA is a fast-paced game where a player controls one robot and battle against another player. However, there has been issues reported by various players in the forum regarding the abuse of network latency in player versus player sessions [27, 68].

Figure 5.1 shows game view of player $A$ and player $B$ from $t(1)$ to $t(3)$. There is no network latency during $t(1)$ and both players have accurate game view. If both players are from the same region, there is a good chance that they can retain their game view consistency. However, problems caused by network latency usually happen when both players come from different region (eg. player $A$ comes from Australia and player $B$ comes from Japan). Still in Figure 5.1, there is latency during $t(2)$ where both players no longer have consistent game view. In $t(3)$, the latency has been resolved and player

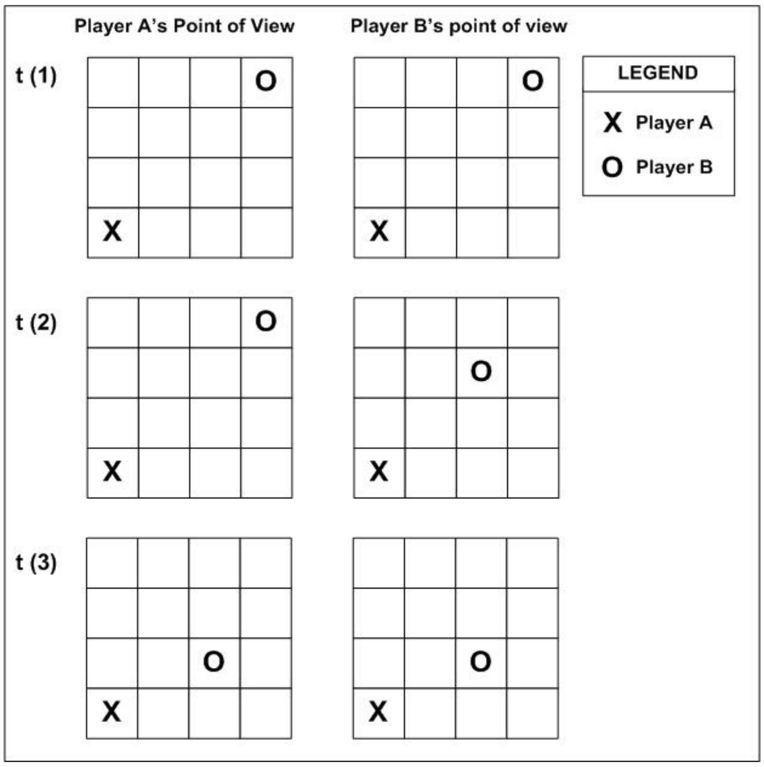

Figure 5.1: Effect of network lag in multiplayer online games

$A$ starts receiving state update from player $B$. From player $B$'s perspective, there is no problem because player $A$ is not moving. However, from player $A$'s perspective, player $B$ 'magically' disappeared during $t(2)$ due to lack of state update and suddenly appears in different location during $t(3)$.

## 5.3 ANN and Dead Reckoning in Multiplayer Online Games

The research about the application of Artificial Neural Network (ANN) for Dead Reckoning is a relatively new area with the most relevant and recent work by [59]. Their research elaborates the implementation of ANN for multiplayer online games using 2D coordinates.

The aim of their research is represented by what they call neuro-reckoning which is basically quite similar with what we want to achieve in our research. They tested their model by deploying their multiplayer games and collected data from players during game sessions. The conclusion of their research supports the initial hypothesis that ANN has the potential to provide better accuracy in dead reckoning for multiplayer games.

The main contributions of our research are the implementation in 3D environment instead of 2D and we are using different type of ANN which we believe that it might have more potential to be implemented in various type of games. [59] uses Multi-Layer Perceptron (MLP) ANN which generally gives good result in general applications. However, there are limitations in flexibility and the ability to adapt with different scenarios. As stated in their work, the ANN can achieve better result by using the training data from the same player. This is due to the fact that one person tend to use the same or similar behavioral pattern. Therefore, the research aim of this chapter is to apply Recurrent Neural Network (RNN) in DR for online games.

Dead Reckoning (DR) is a method to estimate the position of a player based on previous attributes. DR has been commonly used in online games to improve multiplayer gaming experience by reducing the impact of network lag. DR achieves this by providing estimated locations of other players whenever latency occurs so the game will play smoothly in the perspective of players. This becomes more important with the fact that the number of people playing multiplayer online games increases over the years, creating the term Massively Multiplayer Online Games (MMOGs) for the games with a large number of players. Meanwhile, the main drawbacks of DR are the extra processing that takes computation resources, low accuracy in some scenarios and inability to be fully adaptive as explained in [8].

Basically, MMOG is one of the *Distributed Interactive Simulation* (DIS) applications where players are represented by avatars in the game world. The information about game world and everything inside it is what we call Game State. Just like any other DIS applications, it is crucial to keep game state up-to-date and the mechanism to achieve this is highly dependent on network architecture used by the game. In our research we assume the use of Client/Server architecture, which is the most widely used by game service provider. In Client/Server architecture, the global game state is being kept in game server and all players connected to the game server need to send their state update and in return, receive the relevant game state update regularly. This is done periodically by all players connected to the game server.

Commonly, the implementation of DR is to fill the gap between state updates so the player movement will become smooth. It calculates the player's location based on its previous direction and speed, use them to move the player until the next state update arrives. This method produces adequate result in multiplayer games where supporting network has high bandwidth and low latency and packet loss rate. However, once packet loss or latency occurs, the result of standard DR deteriorates rapidly. Data collected from Internet forum by [40] shows that latency tolerance is a subjective matter. Some people could tolerate 200ms latency while others are annoyed by only 50ms. In our research, we employ ANN to actually predict the value of next state update, and use ordinary DR method to move the player to the predicted coordinate.

Although there are various type of MMOGs, our research specifically aims for 3D First Person Shooter (FPS) games. The reason is that not only this type of game is one of the most popular, but also the concept in FPS games usually applies to several other types of games such as Role-Playing Game (RPG) and some of the sports simulation games. Furthermore, 3D multiplayer games are generally more common these days than 2D multiplayer games.

One of the facts that motivates us to use the ANN for DR is due to the similarity between the ability of ANN to predict future values based on previous/past values and the main objective of DR which is to extrapolate data in order to provide estimation of future values based on past values. However, there are few challenges to implement ANN for DR. First, training process of ANN requires a significant amount of data to train an ANN so it could provide satisfactory result. Secondly, so far there is no definite method to construct ANN that produces optimum result. Finally, ANN requires computation resources of player's machine where in most cases, the available resources are required to process game mechanics. Due to those challenges, we use 'trial and error 'method to find the best ANN design and assume that the computation resource requirement is not an issue.

To better explain our research, we use Figure 5.2 that shows the general processes of our DR method and Table 5.3 that lists all of the notations used in this chapter. In Figure 5.2, the whole process starts with data input which serves as normal input for ANN to produce the output or as the input to train ANN. All ANN needs training before it can produce desirable output and our DR process begin with the input that comes either from real-time data input where the ANN will produce the prediction value as the output or training data input to train the ANN. In our research, we use recorded gameplay sessions of the game called Quake II that we obtained from [72]. Although it is an old game, we choose to use it because of recorded session and game source code availability. Therefore, we can use the available data to assist us to analyze the game processes in a more detailed manner.

| **Notation Symbol** | **Description** |
|---|---|
| $t$ | Simulation time instance, one time instance indicates one tick. |
| $P_i$ | 3D Coordinate of point i. |
| $\phi$ | Yaw angle of a 3D vector. |
| $\theta$ | pitch angle of a 3D vector. |
| $k$ | The number of supplied previous data to ANN which equal to the number of input of an ANN. |
| $a_k$ | The vector containing $k$ inputs for ANN. |
| $b$ | The output of the ANN. |
| $s_t^k$ | Training data input vector at time t with k member. |
| $h_t^k$ | Real time data input vector at time t with k member. |
| $g_j(x)$ | An activation function that receives $x$ as the input for neuron in layer $j$. |
| $f(x)$ | The function that represents RNN. The input $x$ is a vector $a_k$. |
| $E_n(t)$ | Represents the vector containing 3D coordinates of player. $n$ contained in a state update at game time $t$. Therefore, $E_n(t) = \{x_t, y_t, z_t\}$. |
| $\xi$ | Error-rate of ANN. |

Table 5.1: Notation and definition

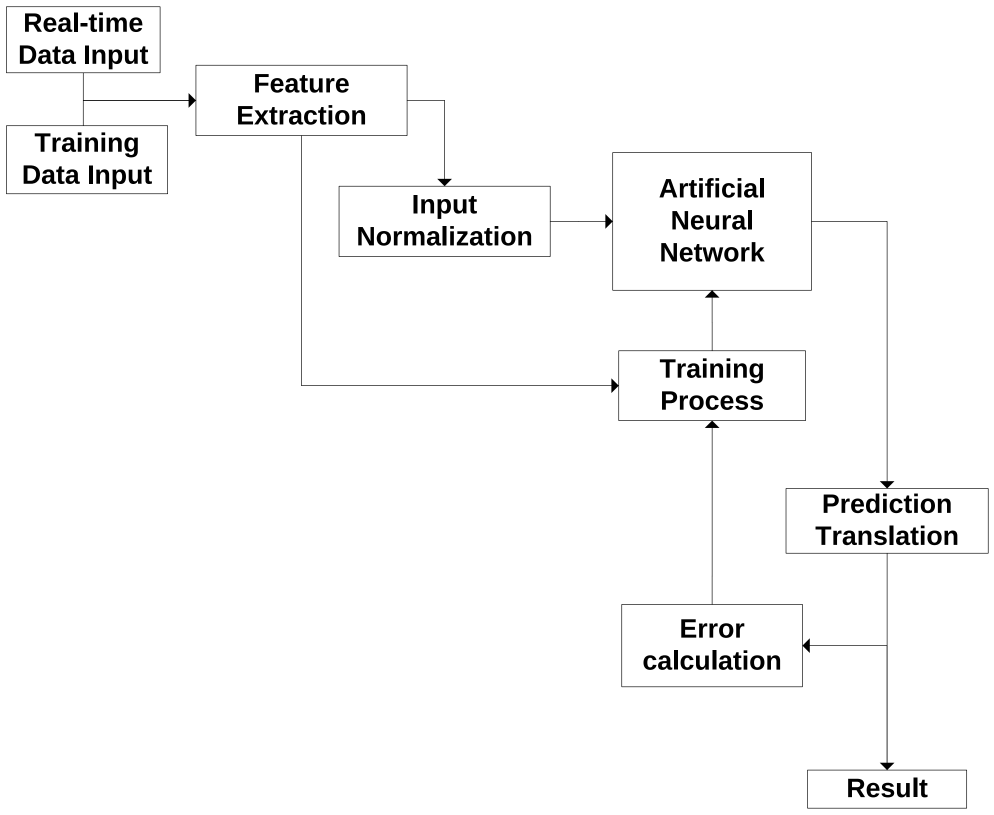

Figure 5.2: General ANN-DR process

Using the same term as in [59], we call the detailed record of player movement from game *Jake2* as High-Fidelity (HF) data and the output of our model and recorded game session from [72] as Low-Fidelity (LF) data. In our model, we use a vector $a_k$ as the input for our ANN. During training process, we read the file of recorded movement data from [72] and put them into a vector. We assume $k = 10$ which means our model requires 10 previous state updates in order to predict the next coordinate. During gameplay, if a player misses more than 1 state update due to latency or packet loss, the current output of ANN can be used as the input to predict the next value as shown in formula (5.1) and

(5.2).

$$b_t = f(a_{t-10}, a_{t-9}, ..., a_{t-1}) \tag{5.1}$$

$$b_{t+1} = \begin{cases} f(a_{t-9}, a_{t-8}, ..., a_t) & \text{if } a_t \text{ is available,} \\ f(a_{t-9}, a_{t-8}, ..., a_{t-1}, b_t) & \text{if } a_t \text{ is not available,} \end{cases} \tag{5.2}$$

In formula 5.1, $b_t$ is the prediction result of the ANN at time $t$. To obtain $b_t$, 10 previous state updates, denoted by $a_{t-10}$ to $a_{t-1}$ are required for the input of the ANN function. In formula 5.2, $b_{t+1}$ is the next predicted state update at $t+1$. If the game server could send $a_t$, then it will become the input for the ANN function. If $a_t$ is not available due to packet loss or network latency, $b_t$ could be used to replace $a_t$. However, the accuracy of the $b_{t+1}$ may be significantly lower. In the next subsection, we will explain all of the processes described in Figure 5.2.

### 5.3.1 Feature Extraction

The performance of ANN is highly related to the forms of input and output. Based on what we get from recorded Quake II game sessions, we obtain player's coordinate in the form of $\{x, y, z\}$ and packet sequence number. Due to packet loss, we encountered missing state updates in every recorded game session file. In this case, we calculate a middle point between two available state updates ($\{x_1, y_1, z_1\}$ and $\{x_2, y_2, z_2\}$) to replace the missing packets by using the following formula:

$$(x_m, y_m, z_m) = (\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}, \frac{z_1 + z_2}{2}) \tag{5.3}$$

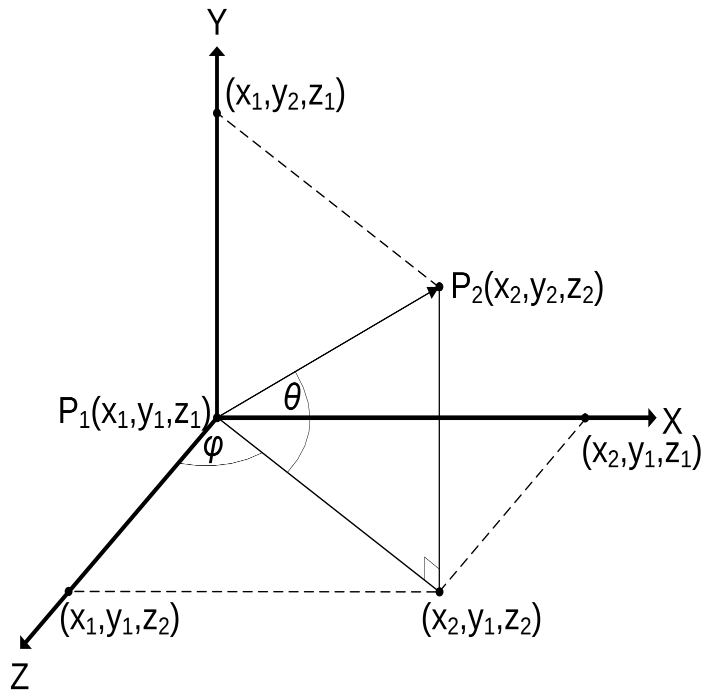

Figure 5.3: 3D Vector direction

In order to provide a pattern, we found that it is not possible to just use the training data as it is. Therefore, we need to extract certain features to reveal the patterns to be recognized by ANN. For example, [59] uses the orientation $(\theta_x, \theta_y)$ and velocity $(v_x, v_y)$ extracted from their high fidelity data. We tried to apply the same method into 3D environment and we found that using three orientation $(\theta_x, \theta_y, \theta_z)$ gave lower DR accuracy. Therefore, we devised a different method to extract features from our data.

The features we are using in our ANN are yaw $(\phi)$, pitch $(\theta)$ and player movement speed which we obtain from player's coordinates. However, we can omit roll because First Person Shooting (FPS) games generally never use roll and set this property to 0. Figure 5.3 illustrates our definition of yaw $(\phi)$ and pitch $(\theta)$. The process to extract the features begin with getting locations of two consecutive state updates of a player, for example $\{x_1, y_1, z_1\})$

and $\{x_2, y_2, z_2\}$. Using these two coordinates, we can calculate the distance ($d$) between those two points using the following formula:

$$d = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{5.4}$$

After we get the distance, we can calculate $\phi$ and $\theta$ by using the following formulas:

$$\phi = atan2(\Delta y, \Delta x) \tag{5.5}$$

$$\theta = \arccos(\frac{\Delta z}{d}) \tag{5.6}$$

Mathematical function $atan2$ in equation (5.5) is a variation of trigonometry function $arctan$. Further detailed information about $atan2$ can be found in [83]. To calculate velocity, we simply use the distance between two coordinates that comes from 2 state update (SU) and make it the speed of the player ($v = d$). Therefore, the measurement unit used for the velocity is point per state update ($\frac{pt}{SU}$). After we obtain yaw, pitch and velocity, we can define vector $P_t(\phi, \theta, v)$ to represent the direction of a player between game time $t$ and $t - 1$.

The next step is the input normalization process. Sometimes, ANN could provide better prediction rate if the input is normalized and we reach the same conclusion after our initial simulations. With data normalization, our ANN could produce better result compared to the one without it. To perform data normalization, we use standard *Max-Min normalization* formula as shown in Formula (5.7) where $I$ is the normalized value and

| **Data type** | $D_{min}$ | $D_{max}$ |
|---|---|---|
| *theta* | 0 | $2\pi$ |
| *phi* | 0 | $\pi$ |
| *v* | 0 | 1500 |

Table 5.2: $D_{min}$ and $D_{max}$ value for $\theta, \phi andv$

$D$ is the original input.

$$I = I_{min} + (I_{max} - I_{min}) * (D - D_{min})/(D_{max} - D_{min}) \tag{5.7}$$

Because the aim is to convert the input value ($D$) into the scale (0..1), then the value of $I_{min}$ is 0 and $I_{max}$ is 1. The value of $D_{Min}$ and $D_{max}$ for our simulation can be found in Table 5.2 . After calculating $I$, the final step of input normalization is to calculate the difference ($\Delta$) between two consecutive rows in all vectors and producing the final vectors for the input of ANN. This is an important step because by making our ANN to predict smaller scale of a variable, we can to minimize the impact of error rate to the game itself. After this step, we can feed the data to train our ANN which we will explain in the next section.

### 5.3.2 ANN Design & Implementation

To provide better adaptive ability to our DR, we choose Recurrent Neural Network (RNN) as our type of ANN. RNN provides better flexibility and also generally better accuracy in prediction because it allows the result to be feed into the ANN in real-time to recalculate node weights. By recalculating the weight as soon as the new result produced by hidden

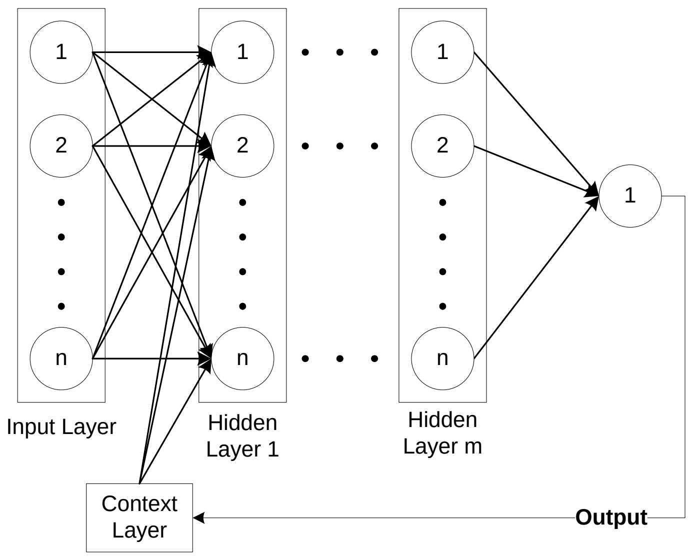

Figure 5.4: Generic Model of RNN

nodes, RNN can theoretically adapt itself to cope with different data set. Our initial references to choose RNN as our type of neural network comes from [48]. Figure 5.4 shows our generic model of RNN.

To assist with neural network design and training, we use Encog Workbench (Java) [39]. However, we use our own code for implementation and evaluation. Every ANN requires activation function for each node and for this purpose, we choose *Hyperbolic Tangent* (tanh) as the activation functions for all nodes of our RNN. Our initial experiments gave poor result when we use one RNN to take the vectors of $\theta, \phi$ and $v$ as the input to produce predicted values. As a reference, three previous experiments gave 21.23333%, 28.99997% and 44.44445% of error-rate ($\xi$) respectively. Therefore, we designed our system that consists of three RNN. Each RNN will take one input vector and we represent each of RNN as $f_\theta(a_k^\theta)$ for $\theta$ prediction, $f_\phi(a_k^\phi)$ for $\phi$ prediction and $f_v(a_k^v)$ for $v$ prediction.

To design our RNN, we use trial and error method to get the best-possible RNN. This includes trying different number of input ($k$), the number of hidden layers, the number

| **Number of input** ($k$) | **Error-rate** ($\xi$) |
|---|---|
| 1 | 55.667% |
| 2 | 46.667% |
| 3 | 41.333% |
| 4 | 44.444% |
| 5 | 42.667% |
| 6 | 38.999% |
| 7 | 32.333% |
| 8 | 18.889% |
| 9 | 10.667% |
| 10 | 4.580% |
| 11 | 7.667% |
| 12 | 9.999% |

Table 5.3: Error rate vs. Number of input

of output and node activation functions. Table 5.3 shows the result of our simulation for $\theta$ prediction with different number of input. The result proves that there is no direct relation between the number of input and error-rate. To the best of our current knowledge, RNN with 10-input seems to give the best result. However, it does not conclude that the possibility of more number of inputs may give better result. The same applies to other factors of ANN such as number of layers and the type of ANN.

Table 5.4 shows the value of $\xi$ in the RNN for $\phi$ with 10 inputs. It shows that 2 hidden layers give the best result. It is possible to get better result using more number of hidden

| **Number of Hidden Layer** | **Error-rate** ($\xi$) |
|---|---|
| 1 | 10.667% |
| 2 | 4.444% |
| 3 | 8.999% |
| 4 | 11.333% |
| 5 | 9.111% |

Table 5.4: Error rate vs. Number of Hidden Layer using RNN with 10 inputs

layer, but in terms of computation resource efficiency, we believe that 2 hidden layers is the best.

In summary of the trial and error process, we found the following results for our RNN design:

- RNN with 10 inputs gives a good result, more than 10 inputs does not give significant improvement.
- $\theta$ and $\phi$ could use the same RNN model (10 input nodes, 2 hidden layer, 1 output) and gives similar error-rate ($\xi$). Meanwhile, RNN to predict $v$ needs one more hidden layer in order to achieve similar $\xi$ during training process.
- More than 3 hidden layers do not give benefit to reduce the value of $\xi$. In fact, our simulation found that the value of $\xi$ will increase when we deploy more than 4 hidden layers.

For training algorithm, we use Genetic Algorithm training method because we found it more stable to find the best result of our RNN with mutation percentage = 0.1 and Percent

| **Epoch** | 20 | 40 | 60 | 80 | 100 |
|---|---|---|---|---|---|
| $\theta$ | 19.99% | 4.68% | 8.89% | 12.67% | 5.67% |
| $\phi$ | 46.67% | 21.99% | 4.44% | 7.67% | 15.33% |
| $v$ | 37.67% | 19.89% | 5.15% | 5.55% | 11.67% |

Table 5.5: Training progress by the number of epoch (Lowest error rate)

to Mate = 0.25. All of our RNN requires a number of previous data to predict future value. To train our network, we extracted data from 20 different recorded game data. From those 20, we have 12 scenarios by the same players, 6 scenarios using the same map but different players and 4 others are by different players and different maps. The best training result in Table 5.5 shows the lowest obtainable $\xi$ against the number of training epoch.

## 5.4 Output De-normalization

The output of an RNN is still in the normalized form. Thus, we need to de-normalize it before we can use to produce predicted coordinate. De-normalization processes are basically the inverse of normalization process. They begin with summing the output of each RNN ($\Delta\theta, \Delta\phi$ and $\Delta v$) with previous corresponded value to get the predicted value of each of them. After getting the value of predicted value, we can translate $\theta, \phi$ and $v$ into $\Delta x, \Delta y$ and $\Delta z$. After this step, we can obtain the final predicted coordinate by adding each of them with $x_{(t-1)}, y_{(t-1)}$ and $z_{(t-1)}$. The whole formula to get the predicted coordinate can be written as follow:

$$x_t = x_{t-1} + ((v_{t-1} + \Delta v)\sin(\theta_{t-1} + \Delta\theta)\cos(\phi_{t-1} + \Delta\phi)) \tag{5.8}$$

$$y_t = y_{t-1} + ((v_{t-1} + \Delta v)\sin(\theta_{t-1} + \Delta\theta)\sin(\phi_{t-1} + \Delta\phi)) \tag{5.9}$$

$$z_t = z_{t-1} + ((v_{t-1} + \Delta v)\cos(\theta_{t-1} + \Delta\theta)) \tag{5.10}$$

## 5.5 Simulation Result & Analysis

From our training processes, the lowest $\xi$ value that we can get for $\theta, \phi$ and $v$ are 4.68%, 4.444% and 5.15%. After training, we use our HF data to test the performance of our RNNs against standart DR formula. Figure 5.5 shows a fraction of DR during a game session. The left-hand side of the figure shows DR using standard formula and the right-hand side of the figure shows DR using our ANN. As we can see in the figure, although our method performs slightly better than standard method, it is still not perfect. We will discuss the factors that could reduce the accuracy at the end of this section.

After our simulation, we can conclude that the error rate of ANN does not have direct relation with the accuracy of the final result. However, $\xi < 5\%$ seems to perform better than general DR method for most situations. We also found that there is a minor correlation between accuracy and players. After training our RNN using the same player in 12 scenarios, the same RNN perform differently if we use it for different players. There is no fixed tendency of being always better or worse. In order to measure the accuracy of our method, we use the distance between predicted coordinate and coordinate from HF data using the following formula:

$$\rho = \frac{\sqrt{(x_p - x_h)^2 + (y_p - y_h)^2 + (z_p - z_h)^2}}{T_{Max}} \tag{5.11}$$

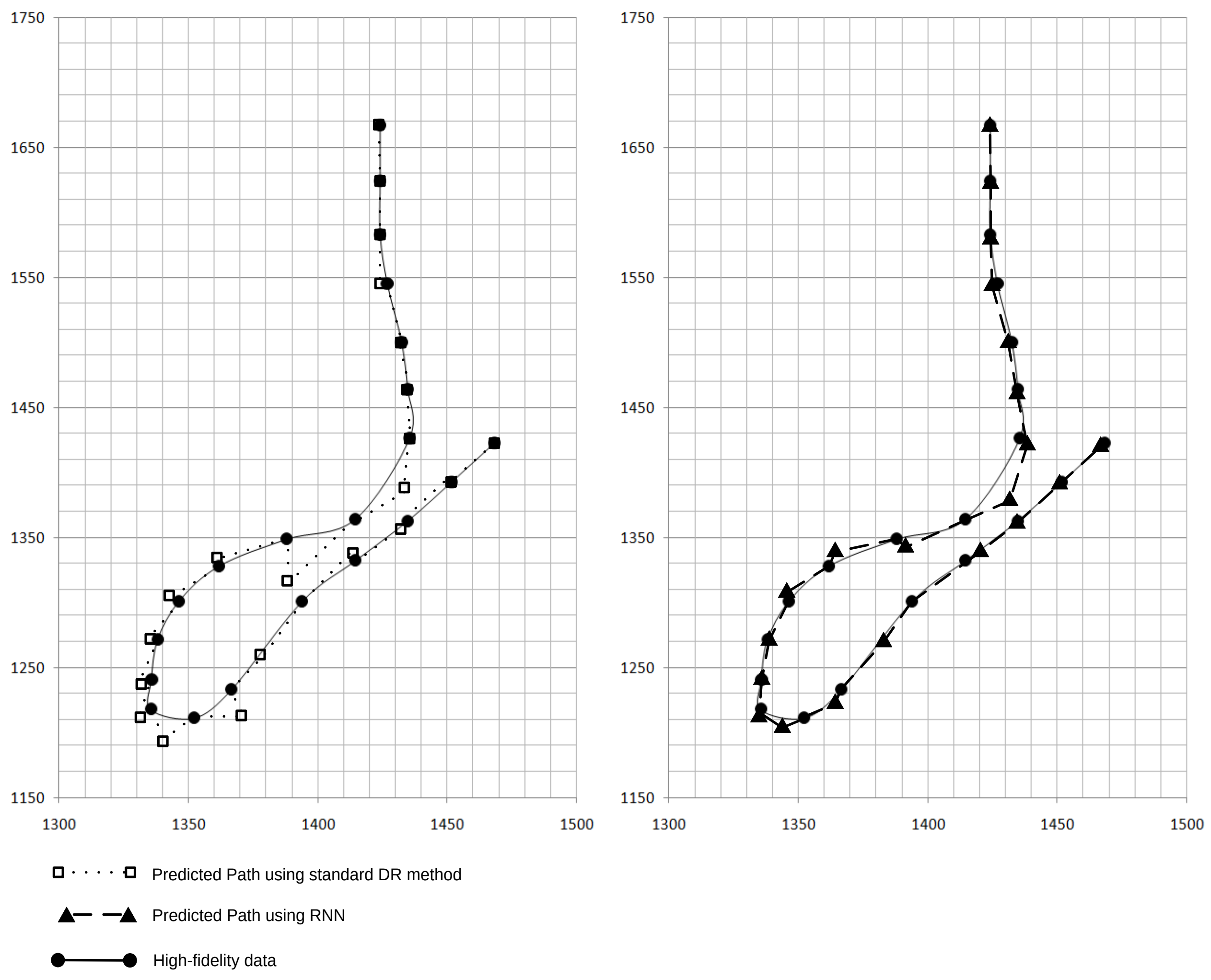

Figure 5.5: DR using Standard Formula and RNN

Where $(x_p, y_p, z_p)$ is the predicted coordinate and $(x_h, y_h, z_h)$ is the coordinate obtained from HF data. $T_{Max}$ is the maximum threshold of how far is the distance between predicted and HF coordinates can be tolerated. In our simulation, we set $T_{Max} = 50$. When $\rho < 1$, it means that the predicted coordinate is still usable. We use $\rho$ in two ways: first, we calculate the average value of $\rho$ during a game session and compare it between different DR methods. The second one is to calculate how many times in one game session that $\rho > 1$. The table below shows $\rho$ calculation result in 10 different game scenarios:

| Epoch | Standard DR | | ANN-DR | |
|---|---|---|---|---|
| | AVERAGE($\rho$) | COUNT($\rho$) | AVERAGE($\rho$) | COUNT($\rho$) |
| Scenario 1 | 0.731 | 1121 | 0.511 | 547 |
| Scenario 2 | 0.695 | 2498 | 0.509 | 1129 |
| Scenario 3 | 0.829 | 1991 | 0.566 | 1215 |
| Scenario 4 | 0.538 | 749 | 0.327 | 823 |
| Scenario 5 | 0.774 | 882 | 0.443 | 545 |
| Scenario 6 | 0.615 | 1294 | 0.520 | 663 |
| Scenario 7 | 0.699 | 1119 | 0.551 | 812 |
| Scenario 8 | 0.514 | 1023 | 0.642 | 841 |
| Scenario 9 | 0.588 | 902 | 0.495 | 910 |
| Scenario 10 | 0.832 | 1885 | 0.789 | 959 |

Table 5.6: $\rho$ result in 10 different Scenarios

As shown in Table 5.6, our method generally performs better than standard DR. Only in scenario 8, the performance of RNN is lower than Standard DR. We believe the cause is large variations in movement pattern produced by different player and different type of maps. Based on the result, we observed our simulation and we found that our RNN produces lower or worst accuracy during certain events in the following conditions:

- When a player makes a temporary circular movement. However, this case is quite rare and our model will adapt to the circular movement given the time.
- When a player makes a small zig-zag movement. This will come up as a straight line using our DR.
- When an external factor suddenly changes the movement direction such as explosion or being hit by a rocket.
- When a player stays in one place for more than 1 second.

Therefore, we decided to include game mechanic calculation into our ANN to improve our result. The involvement of game mechanics process requires the following calculations:

- Collision detection with wall.
- External factor beside $v$ of a player such as jumping, using elevator, using teleporter and other location-related events.
- External force caused by game terrain or other players (e.g. hit by rocket)
- When two state updates confirms that a player stays in the same place, disable ANN prediction until the next state update shows that the player moves to another location.

By implementing four tasks above, we managed to improve $\rho$ by 0.15 in average. However, the implementations itself requires a large computation and memory resources. During testing, we experienced game delay caused by lack of computation resources.

## 5.6 Summary

This chapter presents our DR solution which is based on ANN. Currently, ANN is an intensive and ongoing research field mainly because there is no definite method to find the best ANN design to suit or solve a problem. So far, people usually employ trial and error method to find the best ANN design. This method is time consuming and does not guarantee the best result. In this chapter, we want to show the potential of ANN for DR and for that matter, we produced some results through simulation.

The ANN type chosen in this chapter is RNN. Computation resources required to implement and train RNN in real games prove to be too large for most computers or game consoles at the moment but it is possible to have the calculations done in the server. In this case, game server can take the role to continuously train the RNN during the game session and send the weight value of all nodes to the players periodically.

From simulation, RNN could provide better result than other type of ANN because it could adjust the weight of each nodes in hidden layer to improve accuracy during the game session. Therefore, the next step of our research is to design the ANN that gives the best state update prediction result and to further improve it to achieve lower resource consumption.

Our simulation shows that ANN can achieve slightly better result than standard DR method and we believe that there is a good possibility that other ANN design could perform better. Future research related to gaming application includes cheat detection, player action prediction and better game AI. Also, there is the potential of using ANN for Dead Reckoning to non-gaming applications such as movement pattern analysis in military, aviation and civil engineering.

# Chapter 6

# Artificial Neural Network for Bot Detection System

## 6.1 Introduction

Multiplayer Game is one type of games that allow player to play with other players in the same game. These days, there are multiplayer games that attract a large number of players and they are called Massively Multiplayer Online Games (MMOG). While MMOGs share some of the concepts with Single Player Games, difference in number of players creates additional challenges for the implementation and deployment of MMOG. One of them is cheating problem. In single player, players cheat in order to gain advantage over the game itself and there is no other party affected by this act. However, cheating in MMOG affect the balance and fairness. Eventually, it would reduce the popularity of an MMOG.

Bot is a popular way of cheating in MMOGs which unlike other types of cheating, it does not involve the attempt to change game state where it violates game mechanics.

Instead, it is programmed to do specific tasks in an accurate manner. For example, aim bot in First Person Shooter (FPS) games helps cheating player to aim the weapon more accurately even with low eye-hand coordination skill. Thus, novice player can even beat other players with higher skill. Cheating using bot is more difficult to detect compared to other types of cheating because it does not make illegal changes according to most of game mechanics.

In this chapter, we propose a possible solution to detect bot in MMOG by using Artificial Neural Network (ANN). One of the most common applications of ANN is pattern recognition. Unlike other current solutions, ANN requires training before it could be deployed. For training, we simulated two bot tasks: player movement and action. We found that given sufficient training data, ANN could theoretically recognize a bot in MMOG by checking its pattern of movement or action. In practice, our bot detection system can suggest potential cheating players in MMOGs and leave further judgment to human administrator.

## 6.2 Discussion

In a behavioral study conducted by [21], cheating can be considered as having big negative influence in in multiplayer online games. There are many ways to cheat in MMOGs as explained and classified by [92], which includes the use of bot. While using bot is just one way to cheat in MMOGs, it is very difficult to detect because bot itself is programmed to imitate human player. Thus, bot does not change game data or tampering with game binary. Instead, it merely provides the input to the player avatar just like a player gives the input. In order to achieve this, a cheating player needs to run an external program along with the game. It is possible to counter the cheat by detecting if such programs are running in the player's PC. *Punkbuster* [45], *nProtect* [46] and *Valve Anti-Cheat System* [23] are examples of such countermeasures programs.

Some of the research in bot detection system have various suggestions about how to detect bot in MMOGs. CAPTCHA is one of the most common method to verify human player displaying a picture containing characters during the game. All players in the game need to prove their identity by inputting the correct characters according to the picture. The players giving wrong answers or not responding will be banned by the game server. The first CAPTCHA system was soon becoming obsolete because bot programmers started to implement character recognition system in their bot, effectively bypassing the CAPTCHA system. Research by [99, 36, 99] suggest variations in CAPTCHA system in order to improve it and making CAPTCHA more difficult for bot to evade or trick. However, CAPTCHA can be implemented only in MMOGs which has turn-based mechanism. For real-time games, it is almost impossible to implement CAPTCHA because it will interrupt and annoy the players.

In summary, examples of other methods of detecting bot in MMOGs include: [77] recognizes input pattern generated by computer. They assume that input generated by bot or computer will have different pattern than the input generated by human. As a result, their research has wide application and not limited only to gaming applications but also other non-gaming applications such as online auction. [86] introduces statistic analysis of entity's action in the game to recognize bot. Their result shows that bot performs certain actions more frequently than human player. Similarly, [19] uses network traffic analysis to determine whether the player is a genuine one or a bot. Meanwhile, the use of ANN in cheat detection is still in early stage with one recent work by [34]. They propose the use of ANN to detect cheating in player speed with very low false-positive recognition.

Our mechanism is highly influenced by [59], where they use ANN for dead reckoning. We created our system based on the same principle, but with different input and output processing. Their ANN takes previous coordinates as input and produces predicted coordinate as output. Our system takes previous coordinates as input and produces an index that signify the possibility of bot usage.

## 6.3 ANN for Cheating Detection

We begin with our hypothesis that ANN has the ability to recognize patterns according to the training dataset. This feature of ANN can be used to recognize certain pattern produced by bot. Currently, there are no definite method to determine the best ANN structure to recognize patterns. Most of the design in ANN are based on trial and error method. In our system, the key points to obtain better results using ANN are the choice of features extracted from training data, activation function and input normalization.

To obtain the input of ANN, first we look into the category of bot based on the activity. As mentioned in Introduction section, there are two main activities of a bot in MMOGs: moving (Type 1) and action (Type 2). For movement type, we generated dummy data of bot movement that follows waypoints in game map. One example of this kind of bot is the one in FPS game called *Counter Strike*. In order for a bot to work properly in Counter Strike, it needs to know where to go and which location is more strategic than others. These kind of information is what they called waypoints. In practice, bot will favor certain waypoints and this is what we want to use to recognize bot. Another information that we use in our ANN is player movement pattern. Generally, bot as a machine-based program

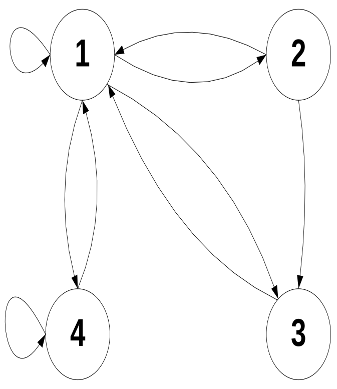

Figure 6.1: Bot Action State

will move in different pattern than human, even with the same destination in game map. Therefore, we choose to use spatial properties of a player such as its coordinates $(x, y, z)$ and movement speed $(v)$.

For action type, we observe a general pattern of action performed by a bot. In the game *World of Warcraft*, a bot might follow the same pattern for its action. Starting with patrolling in one specific areas, the scripted action will determine whether the bot will decide to run or engage an enemy when it encounters one. One way to describe those actions is by using state machine as shown in Figure 6.1. Each state represent an action where state 1 is the idle state, state 2 is the state where a player is engaging the enemy. State 3 is the state where a player is fleeing from an enemy and state 4 is the state where a player is moving around in the game world. In this chapter, we use our ANN to take the sequence of states and determine whether the sequence was generated by human or a bot. Our initial hypothesis that motivates us is the assumptions that human cannot maintain the same sequence for a long time and human player has more random factor when it comes to decision making compared to a bot that always follow a programmed pattern. Therefore, when the accuracy of ANN reaches certain level, it means that a

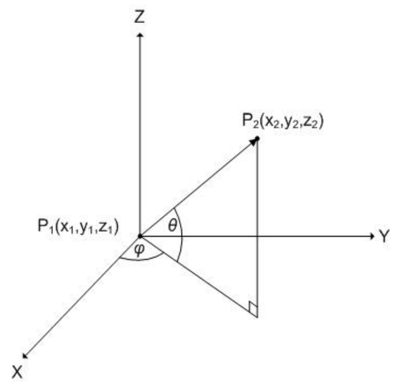

Figure 6.2: 3D Vector direction

player's behavior is suspicious and the player could be a bot.

We obtain our training data for Type 1 by extracting them from recorded bot movement of the game *Quake II*. Our reason to choose this game is due to source code availability. We modified the source code to log the bot movement into a file. With the recorded data, we get the sequence of bot movement coordinates and the features we extract is the direction vector that consist of yaw ($\phi$), pitch ($\theta$) as described in Figure 6.2 and bot movement speed ($v$). First, we need to calculate the distance between two coordinates by using formula 6.1. After we get the distance, we can use it to calculate yaw and pitch (Formula 6.2 and 6.3). Next, we put the results into a vector $a_t^1 = (\phi, \theta, v)$ where t is the state update sequence number.

$$d = \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} \tag{6.1}$$

$$\phi = atan2(\Delta y, \Delta x) \tag{6.2}$$

$$\theta = \arccos(\frac{\Delta z}{d}) \tag{6.3}$$

| | **Type 1** | **Type 2** |
|---|---|---|
| ANN Architecture | Multi-Layer Perceptron (MLN) | Multi-Layer Perceptron (MLN) |
| ANN Topology | 30-15-1 | 12-6-1 |
| Learning Algorithm | Resilient Propagation | Resilient Propagation |
| Node Activation Function | Hyperbolic Tangent (TANH) | Logistic Sigmoid (LOGSIG) |
| Error function | MSE | MSE |
| Max. Step | 50 | 50 |
| Initial Update | 0.1 | 0.1 |
| Training Epochs | 1000 | 1000 |

Table 6.1: ANN Configuration Type 1 and Type 2

For Type 2, we simulate the input data based on the state machine shown in Figure 6.1. We programmed our bot action simulator to produce random sequence with probability for state 1-2-3-4 as 15%-20%-5%-60%. The result of simulator is a vector $a_t^2 = (x)$, where $x = \{1, 2, 3, 4\}$. For the configuration of ANN, we created two different configurations based on trial and error method. The ANN configuration of each type is shown in Table 6.1.

The output for ANN Type 1 ($\lambda_1$) and ANN Type 2 ($\lambda_2$) are what we call confidence level and their value is between 0 to 1. The value 0 signifies big error and it may indicate that the current input is generated by human. On the other side, when the output value is closer to 1, it means the recognition error is low and that could indicate that the current input is generated by a bot. During ANN training for Type 1, we found that one training data of one type of bot cannot be applied to other bots. If applied, it generally gives a very big error which translates into poor bot recognition and detection. We will discuss this further in the next section. Meanwhile, Figure 6.3 describes the general process of our bot detection system. The process is just one part of the global game loop and this process is running in the game server. In every game loop, the game server will receive state update from players. Bot detection system will extract features from state update and feed them into ANN. In our ANN configuration, type 1 requires 30 inputs which translates as 31 state updates. We can write the formula of our ANN as:

$$\lambda(t) = f(a_t..a_{t-k+1}) \tag{6.4}$$

Where $f(x)$ is a function that represents the whole ANN and $k$ is the number of input of ANN.

## 6.4 Experiment Result

For ANN Type 1, we experimented with 2 different bot: Gladiator bot[35] and Reaper bot[73]. We use *Gladiator* bot for ANN training and *Reaper* bot for cross-reference. From our experiment, we receive the following result:

- Provided with valid training data and valid input, our ANN Type 1 produces output above 0.8 throughout game session regardless of the fluctuation of $\lambda$. In practice, we can setup a system that gives a warning to the game administrator whenever the output of ANN is above the threshold for a duration of time. This way, we can use the bot detection system as an fully automated bot-monitor and leave further action to the game administrator. One case example is when the bot detection system suggest that player A might be a bot, the game administrator can use different method such

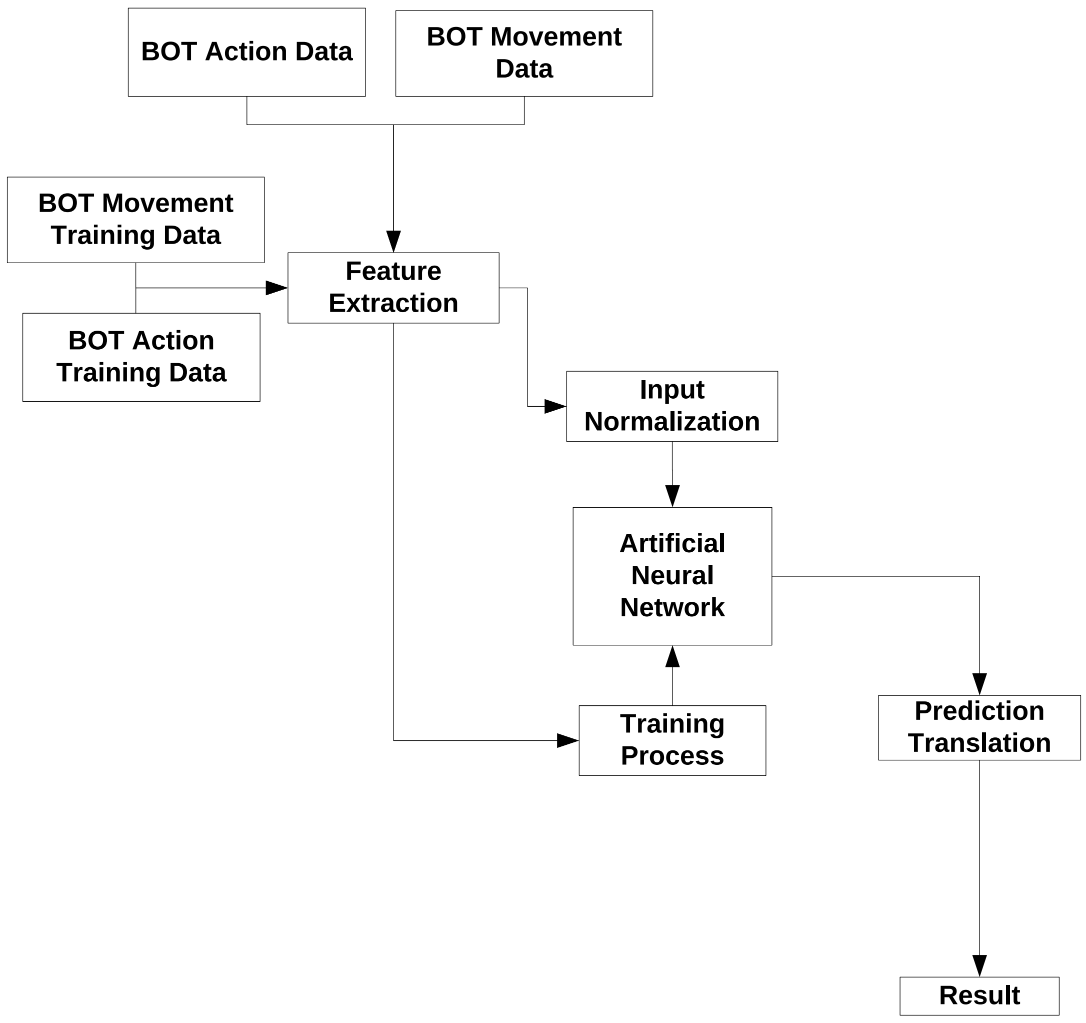

Figure 6.3: General Processes of Bot Detection System

as CAPTCHA to verify player A. If player A failed to verify him/herself, then game administrator can take appropriate measures. If verified, player A can be removed from bot-detection monitor.

- When fully trained using data produced by Gladiator bot, $\lambda_1$ is always below 0.27 if we use the input generated by Reaper bot. Therefore, even though Reaper is a bot, the ANN needs a different separated training and possibly topology in order to get higher $\lambda_1$. The meaning of this result is that one ANN that has been trained to recognize one bot will probably see other type of bots as human. To mitigate this problem, we need different ANN for different bot. It is possible for game provider to automate this process whenever a new type of bot comes out and deploy the new ANN alongside the existing ANNs.

- ANN Type 2 returns higher result than Type 1 by producing $\lambda_2$ constantly higher than 0.9. However, we observe similar effect as Type 1 where $\lambda_2$ is significantly lower when we change the percentage of state probability. In one scenario, we changed state probability into 20%-20%-0%-60% and we receive a constant $\lambda_2 < 0.33$ throughout simulation.

- The accuracy of both types degrade significantly during simulated packet loss. When there is only 1 packet loss, it does not affect the output. However, if packet loss is frequent, the disruption in input sequence causes the pattern to break. Thus, the ANN can no longer recognize the pattern and the output $\lambda$ becomes very low.

## 6.5 Summary

To the best of our knowledge, our bot detection system is the first one to implement ANN as its core system to detect the presence of bots in an MMOG. Unlike other methods, our system is basically use the Artificial Intelligence (AI) in the form of ANN to recognize the pattern generated by the other AI (Bots). Based on our experiment result, we conclude that ANN is a highly considerable option when it comes to bot detection in MMOGs. It strictly needs valid and proper training and input data in order to perform in adequate level. Furthermore, latency and packet loss could affect the accuracy significantly. However, even with those problems, ANN could provide a useful suggestion to aid the game administrator to spot cheater. There is no way to overcome cheating problems in MMOGs completely, but our bot detection system could provide additional filter to find the cheater as proven by our experiment.

Furthermore, ANN is still an ongoing research field and it is highly possible to improve the accuracy by using different ANN configuration or different method to extract values. We leave this as one of our future research possibilities. Also, with the experiment of Type 2, we are convinced that ANN could be used to detect other type of cheating. However, different type of cheating requires completely different ANN input model.

# Chapter 7

# Conclusion & Future Work

Our work in this thesis is designed to be practical, efficient and flexible. We used simulator to test our work and whenever applicable, we used real game data.

First, we introduced our version of GWP System. GWP system improves the efficiency of network resource usage and as the result, directly improves scalability of an MMOG. However, there is possible bottleneck to any implementation of cell-based GWP when a player moves across to another cell. In practice, this bottleneck causes significant delay or loading time. Thus making the players who are crossing the cell to wait until they finished downloading data of the new cell. To tackle this problem, we designed a border detection system using AOI. This system solves the problem by giving the player a chance to start downloading the information possible future cell from the game server before the player actually migrates into that cell. Even though the migration system increases network resource usage, it is still considerably lower than other common methods currently implemented in MMOGs. We also proposed brickworks pattern for GWP. It is basically using square-shaped cells to divide the game world. Therefore, it does not consume much computation resources due to the possibility to perform efficient calculations on squares.

Secondly, for network workload evaluation, we proposed a mechanism to calculate network workload based on the network resources consumed during sending and receiving data from and to the network. We found that not only our solution works well to predict network workload of an MMOG, it is also very useful for other research. Other research could benefit from our work by having a standard method to measure efficiency. So far, almost all research that compares their result with others always develop their own ways to measure and thus are prone to subjectivity. By having one uniform and modular mechanism to measure network workload, more detailed analysis can be performed and in the end, it will improve the quality of other research. Game developers can use our method to assist them during game design and programming. Also, our method can benefit game service provider by providing more accurate estimation of network resource requirement for their MMOGs.

Thirdly, to improve the tolerance to network latency, we proposed an ANN-based method. We choose ANN because it has the prediction and learning ability based on given training data. We chose RNN, which is one type of ANN to be implemented in our solution. From simulation, we found improvement on accuracy over standard DR method. In order to get a valid results, we used real game data for training and simulation. Although RNN is proven to be too much for current generation of game machine, the possibility for improvement is very high. This is due to the unpredictable nature of ANN.

The final issue that we discussed is cheating in MMOG. Bot usage is very common and very difficult to detect. It gives the player a significant advantage by boosting the playing skill. Here, we used the same method (ANN) to detect the pattern of the cheating bot. After having significant amount of training with open source bot, our solutions showed promising results that is better than standard DR method. Moreover, our proposed solutions are capable of providing extrapolation in 2D and 3D MMOG. Also, we conclude that ANN has higher flexibility over other DR methods. The reason is because ANN topology can be modified and re-trained to cope with different kind of game system.

With all of the proposed solutions we offered in this thesis, there are plenty possible future works. In GWP system, possible further work is to gain deeper insight and evaluation of game world partitioning using different network architecture and real data from different type of MMOG. Commonly, the number of packets transferred is different between network architectures and we plan to investigate the effect on game world partitioning. Also, it is possible to combine these methods with other methods [52] so our method can support MMOGs in mobile environment. Furthermore, there are open possibilities to implement internal partition in other cell shapes in order to improve their efficiency.

For network workload evaluation, further work is definitely required to make a unified framework for network workload evaluation that can be applied to all kind of games. So far, our solutions still requires the customization of some parts of the processes in order to be able to give accurate results. Also, we would like to test and expand our simulator using data from other type of games which at the moment we cannot get due to limited available data. Furthermore, it is possible to make a reference database of various event definition and data source. These references then can be used by other research to evaluate their work.

The ANN solution for DR and bot detection system also have a research potential to find more efficient and better ANN topology that could produce better results. There are also possibility to start the research about how to create the best ANN topology automatically. The same research topic is currently ongoing research in Artificial Intelligence (AI) field.

As the knowledge in ANN advances, it is also part of the future work to re-visit these two topics again and see whether the new advances could be applied to improve the current research results.

# Appendix A

# Acronyms

| | |
|---|---|
| AI | Artificial Intelligence |
| ANN | Artificial Neural Network |
| AO | Aion Online |
| AOI | Area of Interest |
| BIP | Brickworks with Internal Partitions |
| CCM | Consecutive Cell Migration |
| CCPM | Consecutive Cell Pre-migration |
| CM | Cell Migration |
| CPM | Cell Pre-migration |
| CS | Client-Server |
| ECT | Event-Cost Table |
| ET | Event Table |
| GWP | Game World Partition |

| | |
|---|---|
| HF | High Fidelity |
| HMM | Hidden Markov Model |
| IMS | Internet Management System |
| IP | Internet Protocol |
| LAN | Local Area Network |
| LF | Low Fidelity |
| LOS | Line of Sight |
| ME | Migration Event |
| MOG | Multiplayer Online Game |
| MMOG | Massively Multiplayer Online Game |
| P2P | Peer-to-Peer |
| PC | Personal Computer |
| PME | Pre-migration Event |
| PvP | Player versus Player |
| RNN | Recurrent Neural Network |
| SPG | Single Player Game |
| WoW | World of Warcraft |

# Bibliography

[1] Quake iii arena. http://www.idsoftware.com/games/quake/quake3-arena/, 2010.

[2] Age of Empires III. http://www.ageofempires3.com/. 2007.

[3] D. T. Ahmed and S. Shirmohammadi. A dynamic area of interest management and collaboration model for p2p mmogs. In *DS-RT '08: Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, pages 27–34, Washington, DC, USA, 2008. IEEE Computer Society.

[4] Aion$^{TM}$Online: The Official Aion Fantasy MMORPG Website. http://na.aiononline.com/. 2009.

[5] T. Alexander. *Massively Multiplayer Game Development (Game Development Series)*. Charles River Media, February 2003.

[6] G. Armitage. *Networking and Online Games*, pages 151–172. John Wiley & Sons, 2006.

[7] Armored Core — For Answer - The Video Game — Ubisoft. http://armoredcoreforanswer.us.ubi.com/. 2010.

[8] J. Aronson. Dead reckoning: Latency hiding for networked games. `http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php`.

[9] M. Assiotis and V. Tzanov. A distributed architecture for mmorpg. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 4, New York, NY, USA, 2006. ACM.

[10] H. Backhaus and S. Krause. Voronoi-based adaptive scalable transfer revisited: gain and loss of a voronoi-based peer-to-peer approach for mmog. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 49–54, New York, NY, USA, 2007. ACM.

[11] Captcha Entry Service - Captcha Bypass. `http://www.beatcaptchas.com/`. 2010.

[12] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. The effects of loss and latency on user performance in unreal tournament 2003®. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 144–151, New York, NY, USA, 2004. ACM.

[13] J.-S. Boulanger, J. Kienzle, and C. Verbrugge. Comparing interest management algorithms for massively multiplayer games. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 6, New York, NY, USA, 2006. ACM.

[14] L. D. Brice no, H. J. Siegel, A. A. Maciejewski, Y. Hong, B. Lock, M. N. Teli, F. Wedyan, C. Panaccione, C. Klumph, K. Willman, and C. Zhang. Robust resource

allocation in a massive multiplayer online gaming environment. In *FDG '09: Proceedings of the 4th International Conference on Foundations of Digital Games*, pages 232–239, New York, NY, USA, 2009. ACM.

[15] B. Carrig, D. Denieffe, and J. Murphy. A relative delay minimization scheme for multiplayer gaming in differentiated services networks. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 36, New York, NY, USA, 2006. ACM.

[16] L. Chan, J. Yong, J. Bai, B. Leong, and R. Tan. Hydra: a massively-multiplayer peer-to-peer architecture for the game developer. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 37–42, New York, NY, USA, 2007. ACM.

[17] K.-T. Chen, P. Huang, C.-Y. Huang, and C.-L. Lei. Game traffic analysis: an mmorpg perspective. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 19–24, New York, NY, USA, 2005. ACM.

[18] K.-T. Chen, J.-W. Jiang, P. Huang, H.-H. Chu, C.-L. Lei, and W.-C. Chen. Identifying mmorpg bots: a traffic analysis approach. In *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 4, New York, NY, USA, 2006. ACM.

[19] K.-T. Chen, J.-W. Jiang, P. Huang, H.-H. Chu, C.-L. Lei, and W.-C. Chen. Identifying mmorpg bots: a traffic analysis approach. *EURASIP J. Adv. Signal Process*, 2009:1–22, 2009.

[20] K.-T. Chen, H.-K. K. Pao, and H.-C. Chang. Game bot identification based on manifold learning. In *NetGames '08: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 21–26, New York, NY, USA, 2008. ACM.

[21] D. Clarke and P. R. Duimering. How computer gamers experience the game situation: a behavioral study. *Comput. Entertain.*, 4(3):6, 2006.

[22] M. Claypool and K. Claypool. Latency and player actions in online games. *Commun. ACM*, 49(11):40–45, 2006.

[23] V. Corporation. Valve anti-cheat system (vac). `https://support.steampowered.com/kb_article.php?ref=7849-RADZ-6869`.

[24] Counter Strike: Source on Steam. `http://www.counter-strike.net/`. 2010.

[25] A. El Rhalibi, M. Merabti, and Y. Shen. Aoim in peer-to-peer multiplayer online games. In *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 71, New York, NY, USA, 2006. ACM.

[26] K. Endo, M. Kawahara, and Y. Takahashi. A proposal of encoded computations for distributed massively multiplayer online services. In *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 72, New York, NY, USA, 2006. ACM.

[27] I. Epic Games. Lag, latency, disadvantage and shot leading explanation. `http://forums.epicgames.com/showthread.php?t=646983`.

[28] Everquest II Sentinel's Fate - Massively Multiplayer Role-Playing Game. `http://everquest2.station.sony.com/`. 2010.

[29] L. Fan, H. Taylor, and P. Trinder. Mediator: a design framework for p2p mmogs. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 43–48, New York, NY, USA, 2007. ACM.

[30] S. Ferretti. A synchronization protocol for supporting peer-to-peer multiplayer online games in overlay networks. In *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*, pages 83–94, New York, NY, USA, 2008. ACM.

[31] T. Fritsch, H. Ritter, and J. Schiller. The effect of latency and network limitations on mmorpgs: a field study of everquest2. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–9, New York, NY, USA, 2005. ACM.

[32] T. Fritsch, H. Ritter, and J. Schiller. Can mobile gaming be improved? In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 44, New York, NY, USA, 2006. ACM.

[33] GameSpy: Age of Empires III - Page 3. `http://au.pc.gamespy.com/pc/age-of-empires-iii/659812p3.html`. 2007.

[34] O. B. Gaspareto, D. A. C. Barone, and A. M. Schneider. Neural networks applied to speed cheating detection in online computer games. *International Conference on Natural Computation*, 4:526–529, 2008.

[35] Gladiator Bot - Website. http://botepidemic.no-origin.net/gladiator/index.htm. 2000.

[36] P. Golle and N. Ducheneaut. Keeping bots out of online games. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 262–265, New York, NY, USA, 2005. ACM.

[37] P. Golle and N. Ducheneaut. Preventing bots from playing online games. *Comput. Entertain.*, 3(3):3–3, 2005.

[38] T. Hampel, T. Bopp, and R. Hinn. A peer-to-peer architecture for massive multiplayer online games. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 48, New York, NY, USA, 2006. ACM.

[39] J. Heaton. Encog artificial intelligence framework for java and dotnet. http://www.heatonresearch.com/encog.

[40] T. Henderson and S. Bhatti. Networked games: a qos-sensitive application for qos-insensitive users? In *RIPQoS '03: Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS*, pages 141–147, New York, NY, USA, 2003. ACM.

[41] History of massively multiplayer online games - Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/History_of_massively_multiplayer_online_role-playing_games. Retreived on June 2010.

[42] S.-Y. Hu and G.-M. Liao. Scalable peer-to-peer networked virtual environment. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 129–133, New York, NY, USA, 2004. ACM.

[43] Ieee standard for distributed interactive simulation - application protocols. *IEEE Std 1278.1-1995*, pages –, Mar 1996.

[44] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, pages 116–120, New York, NY, USA, 2004. ACM.

[45] E. B. Inc. Punkbuster online countermeasures. `http://www.evenbalance.com/`.

[46] L. INCA Internet Co. nprotect gameguard. `http://global.nprotect.com/product/gg.php`.

[47] T. Izaiku, S. Yamamoto, Y. Murata, N. Shibata, K. Yasumoto, and M. Ito. Cheat detection for mmorpg on p2p environments. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 46, New York, NY, USA, 2006. ACM.

[48] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications.* CRC Press, Inc., Boca Raton, FL, USA, 1999.

[49] P. Kabus and A. P. Buchmann. Design of a cheat-resistant p2p online gaming system. In *DIMEA '07: Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*, pages 113–120, New York, NY, USA, 2007. ACM.

[50] H.-G. Kim and S.-W. Kim. An improvement of dead reckoning algorithm using kalman filter for minimizing network traffic of 3d on-line games. In *PCM (2)*, pages 676–687, 2005.

[51] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, and E. Yuk. Traffic characteristics of a massively multi-player online role playing game. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–8, New York, NY, USA, 2005. ACM.

[52] M. Kumar M. M, A. Thawani, S. V, and Y. N. Srikant. Analysis of application partitioning for massively multiplayer mobile gaming. In *MOBILWARE '08: Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, pages 1–6, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[53] Latency Compensating Method in Client/Server In-game Protocol Design and Optimization - Valve Developer Community. `http://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization#fnote3`. 2009.

[54] Lineage II - The Chaotic Throne. `http://www.lineage2.com/`. 2010.

[55] L. S. Liu, R. Zimmermann, B. Xiao, and J. Christen. Partypeer: a p2p massively multiplayer online game. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 507–508, New York, NY, USA, 2006. ACM.

[56] F. Lu, S. Parkin, and G. Morgan. Load balancing for massively multiplayer online games. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 1, New York, NY, USA, 2006. ACM.

[57] R. J. Maly. *Comparison of Centralized (Client-Server) and Decentralized (Peer-to-Peer) Networking.* PhD thesis, ETH Zurich, 2003.

[58] Massively multiplayer online game - Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/MMOG`. 2010.

[59] A. Mccoy, T. Ward, S. Mcloone, and D. Delaney. Multistep-ahead neural-network predictors for network traffic reduction in distributed interactive applications. *ACM Trans. Model. Comput. Simul.*, 17(4):16, 2007.

[60] MMOG Active Subscriptions Chart - 200,000+. `http://www.mmogchart.com/Chart1.html`. 2008.

[61] K. L. Morse. Interest management in large-scale distributed simulations, 1996.

[62] A. Mulholland and T. Hakala. *Programming Multiplayer Games.* Wordware Publishing Inc., Plano, TX, USA, 2004.

[63] J. Müller and S. GORLATCH. Rokkatan: scaling an rts game design to the massively multiplayer realm. *Comput. Entertain.*, 4(3):11, 2006.

[64] J. Müller, S. Gorlatch, T. Schröter, and S. Fischer. Scaling multiplayer online games using proxy-server replication: a case study of quake 2. In *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, pages 219–220, New York, NY, USA, 2007. ACM.

[65] Multiplayer video game - Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Multiplayer_video_game`. Retreived on June 2010.

[66] V. Narayanasamy and K. W. Wong. Mapping resources to state functions in massively multiplayer online games. In *DIMEA '07: Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*, pages 153–160, New York, NY, USA, 2007. ACM.

[67] NCsoft responds to Aion's server queues. `http://www.massively.com/2009/09/23/ncsoft-responds-to-aions-server-queues/`. 2009.

[68] Network lag kills this game at times. `http://forums.modernwarfare247.com/index.php?/topic/11956-network-lag-kills-this-game-at-times/`.

[69] W. Palant, C. Griwodz, and P. Halvorsen. Evaluating dead reckoning variations with a multi-player game simulator. In *NOSSDAV '06: Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, pages 1–6, New York, NY, USA, 2006. ACM.

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

[71] L. Pantel and L. C. Wolf. On the suitability of dead reckoning schemes for games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 79–84, New York, NY, USA, 2002. ACM.

[72] Q2Scene DemoSquad. `http://q2scene.net/ds/`. 2010.

[73] Reaper Bot. `http://easttown.co.uk/quake/reaper.html`. 2010.

[74] S. Rooney, D. Bauer, and R. Deydier. A federated peer-to-peer network game architecture. *IEEE Commun. Mag.*, 42(5), 2004.

[75] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, pages 329–350, London, UK, 2001. Springer-Verlag.

[76] C. Schaefer, T. Enderes, H. Ritter, and M. Zitterbart. Subjective quality assessment for multiplayer real-time games. In *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*, pages 74–78, New York, NY, USA, 2002. ACM.

[77] T. Schluessler, S. Goglin, and E. Johnson. Is a bot at the controls?: Detecting input data attacks. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games*, pages 1–6, New York, NY, USA, 2007. ACM.

[78] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. The effect of latency on user performance in warcraft iii. In *NetGames '03: Proceedings of the 2nd workshop on Network and system support for games*, pages 3–14, New York, NY, USA, 2003. ACM.

[79] B. Software. Jake 2. `http://bytonic.de/html/jake2.html`.

[80] Source Multiplayer Networking. `http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking`. 2009.

[81] A. St. John and B. N. Levine. Supporting p2p gaming when players have heterogeneous resources. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 1–6, New York, NY, USA, 2005. ACM.

[82] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160, New York, NY, USA, 2001. ACM.

[83] I. Sun Microsystem. Math (java platform se 6). `http://java.sun.com/javase/6/docs/api/java/lang/Math.html#atan2(double,double)`.

[84] B. sung Lee, W. Cai, S. J. Turner, and L. Chen. B-s lee et al: Adaptive dead reckoning adaptive dead reckoning algorithms for distributed interactive simulation ? 2008.

[85] M. Suznjevic, M. Matijasevic, and O. Dobrijevic. Action specific massive multiplayer online role playing games traffic analysis: case study of world of warcraft. In *NetGames '08: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*, pages 106–107, New York, NY, USA, 2008. ACM.

[86] R. Thawonmas, Y. Kashifuji, and K.-T. Chen. Detection of mmorpg bots based on behavior analysis. In *ACE '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 91–94, New York, NY, USA, 2008. ACM.

[87] The Challenges of MMOG Development - Page 2. `http://www.digitalgamedeveloper.com/2002/11_nov/features/dlmmogd1126022.htm`. 2009.

[88] The Orange Box - Half Life 2. `http://orange.half-life2.com/hl2.html`. 2009.

[89] T. Triebel, B. Guthier, R. Süselbeck, G. Schiele, and W. Effelsberg. Peer-to-peer infrastructures for games. In *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 123–124, New York, NY, USA, 2008. ACM.

[90] B. Van Den Bossche, T. Verdickt, B. De Vleeschauwer, S. Desmet, S. De Mulder, F. De Turck, B. Dhoedt, and P. Demeester. A platform for dynamic microcell redeployment in massively multiplayer online games. In *NOSSDAV '06: Proceedings of the 2006 international workshop on Network and operating systems support for digital audio and video*, pages 1–6, New York, NY, USA, 2006. ACM.

[91] R. Villamarín-Salomón and J. C. Brustoloni. Bayesian bot detection based on dns traffic similarity. In *SAC '09: Proceedings of the 2009 ACM symposium on Applied Computing*, pages 2035–2041, New York, NY, USA, 2009. ACM.

[92] S. D. Webb and S. Soh. Cheating in networked computer games: a review. In *DIMEA '07: Proceedings of the 2nd international conference on Digital interactive media in entertainment and arts*, pages 105–112, New York, NY, USA, 2007. ACM.

[93] MMOG Active Subscriptions Chart - World of Warcraft. `http://www.mmogchart.com/Chart11.html`. 2008.

[94] World of Warcraft Community Site. `http://www.worldofwarcraft.com/index.xml`. 2010.

[95] World of Warcraft Hits 11.5 Million Users. `http://www.wired.com/gamelife/2008/12/world-of-warc-1/`. 2008.

[96] Z. D. Wu. Performance modelling of multicast groups for multiplayer games in peer-to-peer networks. In *DS-RT '05: Proceedings of the 9th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, pages 105–112, Washington, DC, USA, 2005. IEEE Computer Society.

[97] Z. D. Wu. Modelling for a federated peer-to-peer mmog architectures. *International Journal of Computers and Applications*, 30(4), 2008.

[98] Z. D. Wu. A framework for inconsistency analysis and control of massively multiplayer online games. *Journal of Communications*, 4(10), 2009.

[99] R. V. Yampolskiy and V. Govindaraju. Embedded noninteractive continuous bot detection. *Comput. Entertain.*, 5(4):1–11, 2007.

[100] L. Yang and P. Sutinrerk. Mirrored arbiter architecture: a network architecture for large scale multiplayer games. In *SCSC: Proceedings of the 2007 summer computer simulation conference*, pages 709–716, San Diego, CA, USA, 2007. Society for Computer Simulation International.

[101] T. Yasui, Y. Ishibashi, and T. Ikedo. Influences of network latency and packet loss on consistency in networked racing games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–8, New York, NY, USA, 2005. ACM.

[102] A. P. Yu and S. T. Vuong. Mopar: a mobile peer-to-peer overlay architecture for interest management of massively multiplayer online games. In *NOSSDAV '05: Proceedings of the international workshop on Network and operating systems support for digital audio and video*, pages 99–104, New York, NY, USA, 2005. ACM.

[103] S. Zander, D. Kennedy, and G. Armitage. Dissecting server-discovery traffic patterns generated by multiplayer first person shooter games. In *NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–12, New York, NY, USA, 2005. ACM.

[104] S. Zander, I. Leeder, and G. Armitage. Achieving fairness in multiplayer network games through automated latency balancing. In *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 117–124, New York, NY, USA, 2005. ACM.

[105] Y. Zhang, L. Chen, and G. Chen. Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games. In *NetGames '06: Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, page 7, New York, NY, USA, 2006. ACM.