The effect of delay on network games

Haishu Zhang

May 2, 2006
Master’s Thesis in Computing Science, 10 credits
Supervisor at CS-UmU: Greger Wikstrand
Examiner: Per Lindström

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

Delay of the network-based multi-player games on the Internet is a well-known problem to players. In this thesis a networked Pong game was developed to examine players’ performance for delay.

In order to test how delay affects players, a synchronization mechanism called "dead reckoning" was implemented. Dead reckoning can provide delay feedback by predicting the position of the object, which can reduce the effect of delay. In the experiment the delay feedback was shown as a "shadow" tracking the paddle and a non-feedback condition was also implemented.

The experiment showed that higher delay decreased players’ enjoyment of the game and increased players’ mental effort—the game was more difficult and less fun. If the player receives delay feedback, the acceptance to delay and the mental effort that the player spends should be better.
## Contents

1 **Introduction** .................. 3  
   1.1 Motivation  .................. 3  
   1.2 Structure  .................. 3  

2 **Related work** ................. 5  
   2.1 User performance  .......... 5  
      2.1.1 Network architecture  5  
      2.1.2 Experimental studies  5  
   2.2 Network traffic ............... 6  
   2.3 Network architecture ........ 7  
      2.3.1 Client-Server architecture 7  
      2.3.2 Peer-to-Peer architecture 7  
      2.3.3 Peer-to-Peer with Central Arbiter architecture 7  
      2.3.4 Mirrored-Server architecture 8  
   2.4 Synchronization mechanisms 9  
      2.4.1 Dead reckoning  .......... 9  
      2.4.2 Time Bucket Synchronization 9  
      2.4.3 Time Warp Synchronization 9  
      2.4.4 Breathing Time Bucket Synchronization 10  

3 **Design process** ............... 11  
   3.1 Network architecture ........ 11  
   3.2 Dead reckoning algorithm .... 12  
   3.3 Implementation ............... 12  
   3.4 Delay decorator .............. 13  
   3.5 Software architecture ........ 14  
      3.5.1 Programming language 14  
      3.5.2 Platform ................. 14  
   3.6 Inconsistency ................. 14  
   3.7 Creating transition path .... 15  
      3.7.1 Point-to-Point method 15  
      3.7.2 Linear method .......... 15  
      3.7.3 Quadratic method ....... 15  
      3.7.4 Cubic splines method 15
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Result</td>
<td>17</td>
</tr>
<tr>
<td>4.1</td>
<td>Experiment</td>
<td>17</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Introduced delay</td>
<td>17</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Process</td>
<td>17</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Problem</td>
<td>18</td>
</tr>
<tr>
<td>4.2</td>
<td>Experimental result</td>
<td>18</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Enjoyment</td>
<td>18</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Mental Effort</td>
<td>18</td>
</tr>
<tr>
<td>4.3</td>
<td>Discussion</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>Summary and future work</td>
<td>21</td>
</tr>
<tr>
<td>5.1</td>
<td>Summary</td>
<td>21</td>
</tr>
<tr>
<td>5.2</td>
<td>Future work</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Acknowledgements</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td>25</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Network architecture of the Warcraft III experiment</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Client-Server and Peer-to-Peer architectures</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>Mirrored-Server architecture</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Dead reckoning</td>
<td>12</td>
</tr>
<tr>
<td>3.2</td>
<td>Typical real time multi-player Pong game</td>
<td>13</td>
</tr>
<tr>
<td>3.3</td>
<td>System architecture</td>
<td>13</td>
</tr>
<tr>
<td>3.4</td>
<td>A decorated telepointer shows future states</td>
<td>13</td>
</tr>
<tr>
<td>3.5</td>
<td>Inconsistency in game play caused by network delay</td>
<td>14</td>
</tr>
<tr>
<td>3.6</td>
<td>Four methods to create transition path</td>
<td>15</td>
</tr>
<tr>
<td>3.7</td>
<td>Final results of Cubic splines</td>
<td>16</td>
</tr>
<tr>
<td>4.1</td>
<td>Balanced latin square sequence</td>
<td>17</td>
</tr>
<tr>
<td>4.2</td>
<td>The shadow of the paddle on the left. Shows the predicted position</td>
<td>18</td>
</tr>
<tr>
<td>4.3</td>
<td>Efforts on fun</td>
<td>18</td>
</tr>
<tr>
<td>4.4</td>
<td>Interaction between delay and feedback on mental effect</td>
<td>19</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

As Aronson (J. Aronson, 1997) concluded, delay is not a problem for slow moving games such as chess through the network. In other words, high-speed games such as FPS (First Person Shoot) and Pong can be degraded heavily from delay. Players expect that the performance of the network games can approximate single computer, single player games. Some kinds of prediction methods are tried to achieve this expectancy. With these methods, the next event such as a position update of the object can be predicted (T. Henderson, 2003). These kinds of methods will be discussed in Chapter 2.

1.2 Structure

The outline of this paper is as follows:

- *Chapter 2*: Introduces the related work. Some network architectures and synchronization methods will be introduced;
- *Chapter 3*: Introduces how to use dead reckoning to induce delay and then discuss the inconsistency problem. Later cubic splines methods will be discussed to create a smooth transition between two data points;
- *Chapter 4*: Shows the result of the experiment;
- *Chapter 5*: Conclusion.
Chapter 2

Related work

2.1 User performance

Understanding the effect of delay on user performance in network games is important in order to design networks that meet the requirements. Sheldon et al (N.Sheldor & E.Aqua, 2003) investigated the effect of delay on user performance in Warcraft III, a popular Real Time Strategy (RTS) game.

2.1.1 Network architecture

Warcraft III was divided into its fundamental components of build, explore and combat and experimented separately to measure the effect of delay on each component. Figure 2.1 shows the network architecture that was used to test the game.

![Diagram of network architecture](image)

Figure 2.1: Network architecture of the Warcraft III experiment

Computer A connected the Warcraft computers B, C and D. NIST Net\(^1\) was installed on computer A. NIST Net can emulate a wide variety of network conditions by controlling at the IP level, including fine tuning of delay and variation in delay. NIST Net in this experiment was used to induce delay for one of the computers in the game while others played without induced delay. Ethereal\(^2\) was run to capture packet traces of Warcraft III for network analysis.

\(^1\)http://www-x.antd.nist.gov/nistnet/
\(^2\)http://www.ethereal.com/
2.1.2 Experimental studies

The experiments consisted of two-Play games. One player was experienced an increasing amount of delay ranging from 0 to 3500ms while the other player experienced none. The experiments showed that both players saw exactly the same events on each screen, except the player with added delay saw events later than the player without added delay. From analyzing the experimental data at application level, network level and user level, they found that overall user performance is not significantly affected on the outcome of the game by even very high delay. These results can be explained that RTS games play emphasize on strategy more than the interactive aspects. Since RTS games’ user strategies usually take seconds or even minutes to carry out, the effect of network delay (less than a second) will not impact the overall outcome.

2.2 Network traffic

Internet Service Providers (ISPs) are becoming more and more concerned with delay (M.S, 1999). In these years, a number of ISPs have been provided reasonable low delays. In order to provide this service, the ISPs must have the knowledge of game traffic load according to the networks they provided. Traffic models are required that allow to generate a characteristic load for analytical or simulative performance evaluation of networks to support game traffic. The fast-action multi-player game "Counter Strike" was evaluated from a 36 hour LAN party with 50 participants by Färber (Färber, 2002) to build the game traffic models. Färber observed several matches with 8 to 30 active players lasting 30 to 90 minutes each and then presented traffic models for client and server.

In this game, players joined one of two teams and attack or defend against the other team. It was a very fast paced game which a player’s life usually ended within a few minute. Re-starting the match with a new life was not allowed until the next turn with a turn lasting at most 6 minutes. The game traffic followed the transmit cycle described in (M.S, 1999; Färber, 2002): the server sends game state information to each client where packets are read and processed. When the client receives the server packet, it processes the packet, renders the client’s current view on the screen and samples input devices (mouse, keyboard, joystick, etc), then returns update packets with the player’s movement and status information to the server. The game used UDP packets for the exchange of small update information.

For simple modeling this traffic, Färber treated server and client independently and then presented this traffic model on both server and client parts (Färber, 2002). Packet inter-arrivals and size were concerned with modeling in each part. A mathematical description of the extreme distribution was identified by Borella:

\[
F(x) = \exp[-\exp[-\left(\frac{x-a}{b}\right)]]
\]

\[
f(x) = \frac{dF(x)}{dx} = \left(\frac{1}{b}\right) \exp[-\exp[-\left(\frac{x-a}{b}\right)]] \exp[-\left(\frac{x-a}{b}\right)] b > 0 \ (M.S, 1999)
\]

Parameters a and b are respectively correlated with the mode and the variability of the distribution.
According to the formulas that talked above, Färber has found the best parameter values for Quake traffic (Färber, 2002).

As he said, the whole class of fast-action multi-player games can be described with a general traffic model. This traffic model can be used to evaluate QoS (Quality of Service) aspects of networks with respect to games.

2.3 Network architecture

In order to understand how network delay might affect a game, it is useful to understand the causes of delay, and how games attempt to deal with the existence of this delay. The causes of delay can be divided into end-system factors, such as video compression and decompression, packetisation of data, network factors, such as the actual propagation of data packets through the network. The network is found to be the primary cause of delay (Färber, 2002).

Typically, multi-player games are organized based on the Client-Server (CS) architecture, shown in Figure 2.2(a), or the Peer-to-Peer (PP) architecture, shown in Figure 2.2(b).

![Figure 2.2: Client-Server and Peer-to-Peer architectures](E.Cronin & S.Jamin, 2002)

**2.3.1 Client-Server architecture**

In the CS architecture, one computer acts as a server. This server is responsible for maintaining the database of client state. Every message between the clients is transmitted to the server, and then update messages can be re-sent by the server to all the other clients. It is simpler to maintain state consistency, but this CS architecture has one major limitation: the server bandwidth bottleneck. As Pellegrino et al. (J.D.Pellegrino & C.Dovrolis, 2003) mentioned, in CS architecture, as there was a large bandwidth re-
Chapter 2. Related work

2.3.2 Peer-to-Peer architecture

In the PP architecture, there is no central server. Messages are sent directly from each player to all the other players. All the players are responsible for keeping track of the state. It removes the server bandwidth bottleneck\(^3\), but as each player needs to check the consistency between the local state and the state of all the other players, the entire game state should be copied. This architecture requires synchronization between each player to ensure that the copy of the game state is the same. Without synchronization, inconsistencies might be caused between each player due to the network delay and other factors (E.Cronin & S.Jamin, 2002).

2.3.3 Peer-to-Peer with Central Arbiter architecture

The Peer-to-Peer with Central Arbiter (PP-CA) model (J.D.Pellegrino & C.Dovrolis, 2003) attempts to solve these problems. This architecture combines the merits of CS and PP. The game state is updated using the same manner as in PP architecture but without performing consistency checks. The consistency is checked by a central arbiter that receives all updates with a lower bandwidth requirement than the CS architecture. The reason is that the central arbiter only contacts with players when an inconsistency occurs while the server in CS architecture contacts with players during each update period.

"BZFlag"\(^4\) is used by Pellegrino to experimentally measure the bandwidth requirement and consistency of the previous three architectures. It is a free multi-player multi-platform 3D tank battle game. The name stands for Battle Zone capture Flag. The original BZFlag code did not provide state consistency. Player tanks could occupy the same space on the graphical terrain. The game was modified so that the centralized server can resolve inconsistencies that are related to the position of player tanks.

In this game, four players played between each other for 10 minutes, the first 10,000 packets were captured using "tcpdump"\(^5\) to use in the analysis. Tcpdump is used to sniff network packets and make some statistical analysis out of those dumps. All players were located at the same lab so that the network delays between each player were quite small.

Although the PP-CA helps to reduce the bandwidth requirement, it does not reduce its processing load (J.D.Pellegrino & C.Dovrolis, 2003). Just like a server in CS architecture, the central arbiter needs to handle with the entire game state. This means that if the bottleneck is the CPU power of the server rather than the network bandwidth, the PP-CA architecture will not be scalable either. However, according to the analysis, the PP-CA architecture is the best architecture of the three. This architecture has the best features of CS and PP architecture. It uses the simple consistency resolution mechanism of the CS model to solve the inconsistency problem. The bandwidth requirement

\(^3\) Multicast is required for large number of players. If using unicast, it might get more traffic.

\(^4\) http://www.bzflag.org/

\(^5\) http://www-iepm.slac.stanford.edu/monitoring/passive/tcpdump.html
at the central arbiter increases linearly with the number of players which is significantly lower than in CS architecture that increases quadratically (J.D. Pellegrino & C. Dovrolis, 2003).

### 2.4 Synchronization mechanisms

Some kinds of synchronization mechanisms are needed to facilitate the network architecture, including dead reckoning, Time Bucket Synchronization with dead reckoning, Breathing Time Bucket Synchronization and Time Warp Synchronization (E. Cronin & S. Jamin, 2002).

---

6http://www.idsoftware.com/games/quake/quake3-gold/
2.4.1 Dead reckoning
Dead reckoning is the most commonly used method which can predict the movement of the object. It will be discussed in Chapter 3.

2.4.2 Time Bucket Synchronization
Time Bucket Synchronization was implemented in MiMaze\(^7\) (C.Diot, n.d.; E.Hong & K.Kang, n.d.) which exploited dead reckoning to maintain consistency in the game. Time is divided into fixed size of periods with the length T and a bucket is associated with each period (D.K.Hsiao & D.S.Kerr, 1977). This algorithm has the important property of requiring very little overhead for synchronization (J.S.Steinman, 1993). In MiMaze, events are delayed for a time that should be long enough to prevent disordering before being executed. However, if events are lost or arrive later than expected, it does not attempt to detect inconsistencies (E.Cronin & S.Jamin, 2002). A dead reckoning algorithm (E.Hong & K.Kang, n.d.) is used instead of lost events being retransmitted. Dead reckoning is a prediction mechanism which predicts a next position based on the last position and movement patterns.

For a simple game such as MiMaze, the cost of consistency may be acceptable, but with a game like Quake where interactions between players are much more frequent, small inconsistencies are likely to lead to larger divergences between different states (E.Cronin & S.Jamin, 2002).

2.4.3 Time Warp Synchronization
Time Warp is a synchronization mechanism for parallel/distributed simulation (F.Quaglia, 2002), which can detect out-of-order event execution at run time, and uses a rollback mechanism to recover (J.Tsai & R.M.Fujimoto, 1993). It takes a snapshot of the state before executing, and if errors are detected, a rollback will be issued to an earlier state. In a rollback period, the state is first restored to the last recorded snapshot. Then all events between the snapshot and the execution time are re-executed (E.Cronin & S.Jamin, 2002). Anti-messages are then sent out to cancel the previously generated events which become invalid.

As Time Warp assumes that events directly generate new events, the explosion of anti-messages can bog down the network with anti-message processing instead of executing the game. Another limitation of Time Warp Synchronization is that it requires adding checkpoint at every message which costs a large amount of memory (E.Cronin & S.Jamin, 2002).

2.4.4 Breathing Time Bucket Synchronization
Breathing Bucket Synchronization is described by Steinman (J.S.Steinman, 1993) with the attempts to solve the problem of excessive rollbacks in Time Warp. It also uses

---
\(^7\)http://www-sop.inria.fr/rodeo/MiMaze/
rollback, however, unlike Time Warp, anti-messages are never required. In other words, Breathing Time Bucket avoids the anti-message explosion risk.

Like Time Bucket Synchronization, Breathing Time Bucket divides time into small periods. However, these periods do not use a constant length $T$. They adapt to the optimal width, which is determined by the event horizon (J.S. Steinman, 1993). Events beyond the horizon can not be guaranteed to be consistent, so they are not executed. Thus, in each time period, the maximum number of events can be processed. There is no need to worry about whether the time period $T$ is small or large enough to process events as in the Time Bucket Synchronization.
Chapter 3

Design process

3.1 Network architecture

As discussed in Chapter 2.3, with the CS architecture, the server bandwidth requirement increases fast with the number of players (clients) while with the PP architecture, the server bandwidth bottleneck is removed (Multicast is required for large number of players. If using unicast, it gets a lot of traffic.). But in the PP architecture, as each player needs to check the consistency between the local state and the state of all the other players, this architecture requires synchronization between each player to ensure that the game state in each client is the same. Without synchronization, inconsistencies might be caused between each player due to the network delay.

In the CS architecture, it is usually simpler to maintain state consistency. Another benefit of CS architecture is that in most games, clients are not trustworthy, and cheating is serious in network games. If a client were solely responsible for reporting its location, it would be easy to manipulate the client in order to fake this information. In order to reduce cheating, most games have chosen to use CS architecture, and the server acts as an authoritative source of information and maintains consistency for the players (T. Henderson, 2003). In this chapter we will discuss how to use dead reckoning in the CS architecture.

Geoff (G. Howland, 1999) outlines the additional delay created by client-server game. There are many things that can happen during the process of transmitting packets to other players on the internet, it can collide with another packet and need to be re-sent; it can get lost in the shuffle of routers if a normal connection is down or it can be delayed on a busy network or a number of them. Every message is sent from the client to the server and then re-sent by the server to other clients. This adds additional latency of sending messages to other clients. There is a single authoritative copy of the game state kept at the server, so a player may need to wait for an updated copy of this state before they can be sure of their own position. As a result, this architecture requires some form of synchronization between clients to ensure that each copy of the game state is the same. Without synchronization, due to network delay, the states among players would
diverge over time, this divergence leads to inconsistency.

3.2 Dead reckoning algorithm

Players in a client-server game such as Pong are dependent on receiving state update messages from the server in order to know what is happening to other players. It would be very expensive to send updates continuously, so state updates are sent at time intervals. Therefore the clients need a method to approximate the state between receiving these updates, or in the event of packet loss. One of the most commonly used methods is dead reckoning which was also implemented in the experiment.

Dead reckoning is used to reduce the effects of delay in a game by trying to predict a position update that an object takes (J.Aronson, 1997). In another word, the accuracy of the prediction is very important. Dead reckoning predicts the position based on the object’s characteristics. For example, if an object has a known starting position and velocity then its path can be created using simple physical algorithms. The created paths can be applied to the object, creating the illusion of smooth motion (N.Caldwell, 2000).

Figure 3.1: Dead reckoning

(J.Aronson, 1997)

Figure 3.1 shows an example of dead reckoning (J.Aronson, 1997). A paddle starts to move ahead at time $t_0$, the thick solid line is the true path that the controller moves while the dashed line is the trace that all other players on the internet see. Remote computers can show smooth motion based on the dead reckoning algorithm when the dead-reckoned predict position and the true position stay within a predefined threshold. It will be discussed later. However, at time $t_1$, the discrepancy between the controller simulation and other simulations become bigger than the dead reckoning threshold, a new packet is sent out to all the computers immediately and they can use new packet data and reset the dead reckoning algorithms to resynchronize the position of the paddle at time $t_1'$. The value of dead reckoning threshold is dependent on the type of application. (In the experiment, the threshold was set to 5 pixels.) Nick (N.Caldwell, 2000) suggests using cubic splines method to resynchronize the position of the paddle from $t_1$ to $t_1'$ smoothly. This method will be introduced in Section 3.7.
3.3 Implementation

In this section we will explain the specific working flow of how dead reckoning is implemented in CS architecture by using a typical real time multi-player Pong game. In this game the players use paddles to hit the ball between each other. The ball can also bounce when it touches the walls. If the ball moves past a player’s paddle, that player’s opponent scores a point. A screenshot is shown in Figure 3.2.

![Figure 3.2: Typical real time multi-player Pong game](image)

CS architecture is used in this game as the network architecture (see Figure 3.3) because it is usually simpler to maintain state consistency. Two players play the game on separate computers (clients). A third computer is used as server for connection and data transmission.

1. In Figure 3.3, when an object is created by Computer A, it sends out the PDU (Protocol Data Unit) to all other computers (Computer B) on the network. The PDU contains the unique information of the object including position, velocity, acceleration, orientation and so on. It also contains an identifier to tell Computer B which dead reckoning algorithm this object uses.

2. When Computer B receives the PDU, it creates local copies of the object and begins to move the object using dead reckoning algorithm that contained in the PDU, let the object continue to move in a predictable fashion.

3. Computer A remembers the last time it put out a PDU and runs the dead reckoning algorithm based on that PDU so it has a copy of what Computer B is seeing as well as the true, latest value.
4. Computer A compares the dead reckoning values to the true state of the object which is based on the inputs from the mouse controller that controlled by the player. If the discrepancy exceeds the dead reckoning threshold, a new PDU will be sent out to update Computer B.

5. Computer B updates its copies of the object to reflect the new PDU values, and dead reckoning begins again with the new data point.

### 3.4 Delay decorator

Gutwin et al. explored a different approach to dealing with delay (C. Gutwin & Greenhalgh, 2004). They used decorators to reveal the effects of delay to users so that they could plan their activities in advance. According to the decorators that introduced by the authors, one kind of decorators called "future state decorators" (see Figure 3.4) can be implemented in the Pong game. The future state decorators show the predicted future positions of objects.

![Figure 3.4: A decorated telepointer shows future states](C.Gutwin & Greenhalgh, 2004)

In Figure 3.4, the solid telepointer presents the current state while the dashed trail and faded telepointer shows the possible future state. This method can not reduce delay, but it might reduce the effects of delay on the user. In the experiment we introduced "shadow" of the paddle as the future state decorator, which will be discussed in Chapter 4.

### 3.5 Software architecture

#### 3.5.1 Programming language

- Java is a high level programming language that can be implemented on different platforms, also perfect for the web.
- Java supports easy GUI construction.
- Java enables developers to write applications for consumer devices such as cell phones. Today, all PDA devices are Java enabled.
3.5.2 Platform

The system was run on Java enabled platforms. The target Java version is Java 2 Runtime Environment build 1.4.2 with Swing support. This version of Java provides many improvements over the earlier version, including a significant improvement to the GUI event model.

3.6 Inconsistency

Dead reckoning can help to minimize the effects of latency, but it can also introduce problems of its own (G.Wikstrand, 2004). In Figure 3.5, Player A starts moving at time $t_0$ and continues to move until $t_2$. The distance that the player has moved is $2x$. Suppose at $t_1$ when Player A has moved $x$ units, the server receives the information that Player A has started moving. This information is immediately sent to Player B who receives it at time $t_2$. In fact, at this time Player A has already stopped moving. So at the same time that Player A stops moving Player B will see that Player A just starts moving. Player B can not see Player A stop until $t_4$. Network delay can therefore lead to inconsistencies between each player’s states.

![Figure 3.5: Inconsistency in game play caused by network delay](image)

Time Warp algorithm can be used to solve this inconsistency (T.Henderson, 2003). The application makes snapshots of the entire player’s state periodically. When an
inconsistency is detected, a “time warp” is performed to revert the state to the last recorded snapshot. However, keeping snapshots can be memory expensive and some researchers have proposed using multiple servers instead (E. Cronin & S. Jamin, 2002).

3.7 Creating transition path

In Figure 3.1, in order to form a smooth path from the position at time $t_1$ to the position at time $t_1'$, four methods can be used. They are discussed as follows.

![Figure 3.6: Four methods to create transition path](N.Caldwell, 2000)

3.7.1 Point-to-Point method

This is the most basic method among these four methods. As the name says, it jerks a player to a new point when a packet arrives (see Figure 3.6 (a)). This method is by far the worst because object jumps from one position to the next, there is no motion path shown on the screen. Unless the remote player stays still, his screen representation is completely wrong (N. Caldwell, 2000).

3.7.2 Linear method

This method is much improved than the point-to-point method (see Figure 3.6 (b)). It creates a “straight-line” path from one position to the next. The next position is determined by the velocity of the object: $Position_{t_1} = Position_{t_0} + vt$. A serious problem is also contained in this equation: it assumes object will move only with a constant velocity. When using this method, player will move in straight lines and whenever he starts a new linear path his velocity will change abruptly. It will cause the object move unrealistic (N. Caldwell, 2000).
3.7. Creating transition path

3.7.3 Quadratic method

This method (see Figure 3.6 (c)) adds a quadratic function in the straight-line path so as to change the straight line into curve: \( \text{Position}_1 = \text{Position}_0 + vt + \frac{1}{2}at^2 \). This method can represent a player’s motion more realistically. However, it also has problems that quadratic function does not change acceleration over time, so the player’s velocity will still change abruptly. The next method is the final solution to solve this problem thoroughly.

3.7.4 Cubic splines method

This method (see Figure 3.6 (d)) is the most realistic method because the objects which follow cubic splines path will have no jitters, unless lag is especially serious (N.Caldwell, 2000).

1. It needs four \((x, y)\) coordinates as its parameters to calculate. These four parameters are shown as follows (see Figure 3.7):

\[
\begin{align*}
\text{Position}_1 &= \text{StartPosition} \\
\text{Position}_2 &= \text{StartPosition} + v \\
\text{Position}_3 &= \text{EndPosition} - v \\
\text{Position}_4 &= \text{EndPosition}
\end{align*}
\]

In these four equations, the first parameter \(\text{Position}_1\) is the object’s starting position \(\text{StartPosition}\); the second parameter \(\text{Position}_2\) is the position after \(\text{StartPosition}\) moves one second using starting velocity; the third parameter \(\text{Position}_3\) is the position after \(\text{EndPosition}\) moves one second using reversed ending velocity; the forth parameter \(\text{Position}_4\) is the object’s ending position \(\text{EndPosition}\).

2. The coordinates of \((x, y)\) to form the spline path is as follows:

\[
\begin{align*}
x &= At^3 + Bt^2 + Ct + D \\
y &= Et^3 + Ft^3 + Gt + H
\end{align*}
\]

The variable time \(t\) in the equations ranges from 0 to 1. At \(\text{Position}_1\) the time value is 0 while at \(\text{Position}_4\) is 1.

3. Here are the variables from A–H:

\[
\begin{align*}
A &= x_3 - 3x_2 + 3x_1 - x_0 \\
B &= 3x_2 - 6x_1 + 3x_0 \\
C &= 3x_1 - 3x_0 \\
D &= x_0 \\
E &= y_3 - 3y_2 + 3y_1 - y_0 \\
F &= 3y_2 - 6y_1 + 3y_0 \\
G &= 3y_1 - 3y_0
\end{align*}
\]
\[ H = y_0 \]

Once the equation is created above, the next step is to get the value of \( \text{Position}_1 - \text{Position}_4 \).

- \( \text{Position}_1 \) and \( \text{Position}_2 \) are easier to get. They can be found using Start-Position and current velocity.
- \( \text{Position}_3 \) and \( \text{Position}_4 \) are more difficult to get. If any new PDU is arrived between \( \text{Position}_1 \) and \( \text{Position}_4 \), simply define it as NewPacket.

\[
\text{Position}_3 = \text{NewPacket}.x + \text{NewPacket}.y \times t + \frac{1}{2} \text{NewPacket}.a \times t^2
\]
\[
\text{Position}_4 = \text{Position}_3 - (\text{NewPacket}.v + \text{NewPacket}.a \times t)
\]

![Diagram](image)

Figure 3.7: Final results of Cubic splines
(N.Caldwell, 2000)

Then make the object travel along the spline for \( T \) frames. If there is no NewPacket arrived, \( \text{Position}_3 \) and \( \text{Position}_4 \) will resume from the beginning. In Fig.3.7 we can see the final result.
Chapter 4

Result

4.1 Experiment

4.1.1 Introduced delay

Two levels of delay were introduced: 50ms and 250ms. As the same value of delay were used both when sending and receiving data packets, the delay was approximately doubled to 100ms and 500ms.

4.1.2 Process

The experiment process contained sixteen sessions which were divided into four rounds. Four participants attended in the experiment and were divided into two groups. Each group consisted of a pair of participants. After each round the participants were asked questions about what they thought during the process of the game. In order to achieve a counter-balanced design, the sequence was altered to form a balanced latin square shown in Figure 4.1 (G.Wikstrand, 2004).

The experiment was divided into four levels, two of which were delay of 100ms and 500ms without dead reckoning while the rest were with dead reckoning as shown in Figure 4.1. In the levels that used dead reckoning, the predicted position of each participant’s paddle was shown as ”decorator” that has discussed in Chapter 3.4. This kind of decorator was represented as ”shadow” in this experiment to provide delay feedback to participants (see Figure 4.2). In Figure 4.2, the left faded red paddle is the shadow of the right red one. It is the right red paddle’s next predicted position.

With the levels that didn’t use dead reckoning, there was no delay feedback. The participants didn’t know the set of each level so we could compare the results.
4.1.3 Problem

In the experiment, a problem happened during the game session of Level 2 (Delay = 100ms, with dead reckoning). Due to some technical problems (there might be something wrong with sending and receiving data packets), the opponent’s paddle jumped up and down in the game which might confuse the participants and would do some negative effects on the final result.

4.2 Experimental result

There are two aspects that are vital to evaluate the computer games which will be analyzed later, one is player’s enjoyment and the other is the mental effort.
4.2.1 Enjoyment

The "enjoyment" means how much fun the player feels during the game period. Four participants joined the game as two groups and were asked how much they enjoyed each session after the experiment to measure the enjoyment by using a nine-point scale (1–absolutely no fun, 2–almost no fun, 3–a little fun, 4–some fun, 5–rather much fun, 6–considerable fun, 7–great fun, 8–very great fun, 9–extreme fun). The result is shown in Figure 4.3.

![Diagram](image)

(a) Delay

![Diagram](image)

(b) Feedback

Figure 4.3: Efforts on fun

The level of delay is found to have a significant effect on a player's enjoyment. In Figure 4.3(a), the trend shows that players enjoyed low delay (Average = 4.9) better than high delay (Average = 4.3). The enjoyment decreases when adding the feedback of shadow. Figure 4.3(b) shows that players enjoyed the game without feedback of shadow (Average = 4.8) better than with feedback of shadow (Average = 4.4). The players interpreted that the shadow sometimes made them confused and might create a less enjoyable experience for them.
4.2.2 Mental Effort

The result was gathered using Rating Scale Mental Effort (RSME) Applet written by L. Schedin\(^1\) after the experiment. The general idea of RSME is to make the person perform a task to measure how much effort it took for the person to complete the task. The subject is then to rate his mental effort by indicating it on a 150-point scale as follows:

- Extreme effort 112 (or 103)
- Very great effort 102 (or 101)
- Great effort 85
- Considerable effort 71
- Rather much effort 57
- Some effort 38 (or 37)
- A little effort 26
- Almost no effort 13
- Absolutely no effort 2

Each participant was asked to measure the mental effort using RSME after each round. The result is shown in Figure 4.4.

![Figure 4.4: Interaction between delay and feedback on mental effect](image)

In Figure 4.4, it is obvious to see that with and without feedback, the mental effect of the participants both increases when the delay increases. With feedback the mental effort appeared to a little higher and actually almost the same as without feedback. It was interpreted by the participants that it’s difficult for them to get used to the shadow in only a few minutes during the experiment period.

4.3 Discussion

In this chapter I have examined how players believe that they respond to the presence of network delay, and compared this to how they actually react in experiment conditions.

\(^1\)http://www.cs.umu.se/~schedin/rsme/
4.3. Discussion

The result of the experiment shows that network delay is a negative aspect to the game. It makes the participants more difficult to control the game so as to increase the mental effort and reduce the enjoyment of the game.

According to the problem that was mentioned in section 4.1.3, I interviewed the participants after the experiment. The investigation shows the "jumping paddle" problem more or less increased their mental effort, which means, the game became more difficult in a certain extent.

To summarize, in this chapter I have shown the following:

- Higher delay decreases players’ enjoyment of the game
- Higher delay increases players’ mental effort-making the game more difficult
- Feedback doesn’t help that much as expected in higher delay
Chapter 5

Summary and future work

5.1 Summary

Network delay is an important factor in the player’s experience in a networked high-speed game. Understanding and reducing the effects of delay on user performance in network games become an important issue.

In Chapter 2, I have presented some synchronization algorithms to deal with delay while in Chapter 3 I chose ”dead reckoning” algorithm to implement in the experiment. With dead reckoning, the next event such as a position update is predicted. It can not reduce the delay directly, but can be used to reduce its impact. The decorators (feedback) can not reduce delay, but it can reduce the effects of delay on users.

This thesis has attempted to investigate how a player can tolerant for the negative effects of network delay in a real-time Pong game. I divided fundamental user performance into two main aspects-enjoyment and mental effort and investigated separately.

5.2 Future work

The initial supposition for the investigation was that there should be significant main effects of both delay and feedback on enjoyment of playing the game and mental effort respectively. I have been able to conclude that delay does indeed have a significant effect both on a player’s enjoyment and his mental effort. The results approved that when delay increased, the player’s enjoyment decreased and the mental effort increased respectively-the game was more difficult so as to make the player feel less fun. However, the feedback neither helps to increase player’s fun nor decrease game difficulty according to the investigation in the experiment. This does not mean that the initial supposition was incorrect. Some additional methods might be used to make the feedback more natural in order to make the participants less confused with the ”shadow”.

27
Chapter 6

Acknowledgements

I would like to thank Greger Wikstrand, my supervisor, for his many suggestions and constant support during this research.

I am also thankful to Mr. Per Lindström for his guidance through the first year of chaos and confusion.

Professor Jurgen Bötler taught Object-Oriented Software Development and Student Conference courses, which gave me a better knowledge background on my report.

I would also like to thank all my friends who provided their help and friendly encouragement.

Of course, I am grateful to my parents for their patience and love. Without them this work would never have come into existence.

Haishu Zhang
References


J. D. Pellegrino, & C. Dovrolis. (2003). Bandwidth requirement and state consistency in 
three multiplayer game architectures. Network and System Support for Games, 
Proceedings of the 2nd workshop on Network and system support for games, 52-59.

Simulation, Proceedings of the seventh workshop on Parallel and distributed sim-
ulation, 109-118.

programs. Winter Simulation Conference, 697-705.

Conference.

(http://www.gamedev.net/reference/articles/article914.asp)

in warcraft iii. Network and System Support for Games, Proceedings of the 2nd 
workshop on Network and system support for games, 3-14.

doctoral dissertation, University College London.