Improving Response Time in a Client/Server 3D Mobile Game

Prapat Lonapalawong
Dept. of Computer Engineering
Prince of Songkla University
Hat Yai, Songkhla, 90110, Thailand
prapatz@yahoo.com

Andrew Davison
Dept. of Computer Engineering
Prince of Songkla University
Hat Yai, Songkhla, 90110, Thailand
ad@fivedots.coe.psu.ac.th

ABSTRACT
A series of experiments were carried out on a client/server 3D mobile first-person shooter (FPS) to determine the best techniques for improving client-side response times in the presence of severe network unreliability. We utilized three measures of response time, which closely parallel the different types of communication employed between the clients. The response time techniques were grouped into three categories: general, game-specific, packet-based. A combination of the best three – dead reckoning and smoothing, avatar blinking, and duplicate/triplicate packet sending – produce mean response times that are 20% to 90% less than the mean response time for the game with no techniques enabled.

Categories and Subject Descriptors
C.2.4 [**Computer-Communication Networks**]: Distributed Systems – *client/server*.

General Terms
Measurement, Performance, Design, Reliability.

Keywords
Client/server, 3D, mobile game, response time measurement, dead reckoning and smoothing, avatar blinking, duplicate/triplicate packet sending.

1. INTRODUCTION
Interest in multiplayer 3D gaming has never been higher, and is starting to gain traction on mobile devices, with the success of games such as *Robot Alliance* and *Need for Speed: Carbon*. However, underlying networking issues (e.g. high latency, limited bandwidth, and lossy/reordered packet delivery) make it difficult to implement FPS-type games that offer rapid player interaction [1, 2, 6]. As a result, many multiplayer mobile games are turn-based, and use the network primarily for messaging and accessing server-side databases.

CyberGames 2007, September 10–11, 2007, Manchester, UK.

This paper describes experiments carried out upon a client/server 3D mobile FPS. The game executes on a LAN, but the server can simulate varying degrees of communication reliability, thereby emulating WAN/Internet conditions. A range of techniques for improving the game's response time were tested, which fall into three broad groups: general (applicable across a wide-range of FPS games), game-specific (tailored to our game), and packet-based. The success (or otherwise) of the techniques was judged by gathering statistics related to three different measures of response time.

2. GAME ARCHITECTURE
Our game's client/server architecture is quite typical of many multiplayer mobile games. The Java ME (`http://java.sun.com/j2me/`, [5]) game clients each render a world of competing penguins; the goals of a player's penguin are to find "life spots", gather bullets, and shoot other penguins. The game's architecture is summarized in Figure 1.

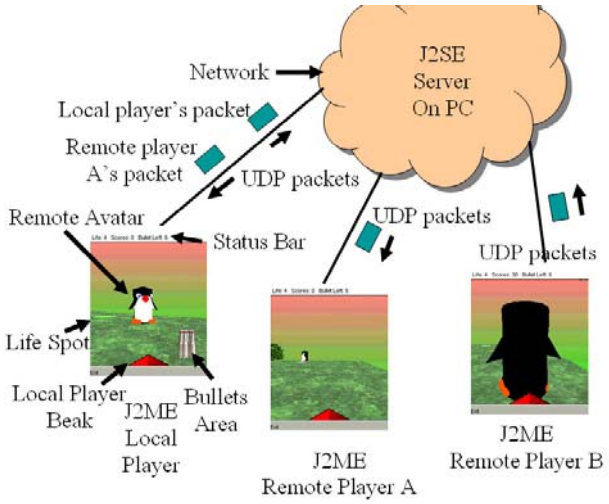

Figure 1. The client/server 3D mobile game.

The local player has a first-person view of a world, while the other penguins are remote avatars representing the other players. In Figure 1, the game currently has three users, so each player can see at most two other penguins (and its own penguin's red beak).

The rules of the game ensure that player behavior is fairly complicated, making it hard to predict a player's actions and the pattern of network activity. All the game's 3D assets (e.g. the penguins, the floor) are stored locally on the clients; no 3D models are transmitted via the network.

Game entry, inter-client communication, and game departure are controlled through a Java SE (`http://java.sun.com/j2se/`) server which manages the delivery of data in the form of UDP packets. The server can be configured to delay packet delivery, and to lose a given percentage of datagrams, in order to test the game's responsiveness at different levels of network reliability. The system was run across a LAN, so real-world latency, bandwidth restrictions, and packet loss were not issues.

Various levels of reliability were investigated, including 75% reliability, which means that there was a 25% chance of a packet being delayed (i.e. one chance in four), and a 25% chance that it would be lost. 90% reliability means that there is a 10% chance of packet delay, and 10% chance of packet loss. A packet can be delayed between 30 ms and 2 seconds.

2.1 Measuring Response Time

A more accurate reflection of a game's responsiveness can be gained by measuring three slightly different forms of response time: one-way response time for *single* packet actions, one-way response time for *multiple* packet actions, and two-way response time.

One-way response time for an action is the time that a packet representing the action takes to travel from a remote player to the local player, and includes the time to update the remote player's avatar on the local device.

Some complicated types of action require multiple packets to be transmitted, typically for updating avatar position and orientation. However, most actions can be represented by single packets, such as when the player loses a life point or picks up a bullet. This distinction between multiple and single packets is important since it highlights the effectiveness of techniques which group, delete, or duplicate packets.

Two-way response time is the time for a packet to be sent from the local player to a remote device to be processed, and for a response packet to arrive back at the local player and update his game state. An example of two-way response time in our game is when a player shoots at a penguin. This requires that a message be sent to the remote client represented by the penguin, and for the local client to wait until the shot's outcome (e.g. penguin death) is returned.

3. TECHNIQUES FOR IMPROVING RESPONSE TIMES

We experimented with a large number of techniques to improve the game's response times. We classify these techniques into three groups:

1. *General* techniques, which can be applied to any networked FPS. They include dead reckoning and smoothing, and selective visual field updating [3].

2. *Game-specific* techniques, which include avatar blinking and avatar dying (i.e. painting a translucent skull over a penguin to indicate its probable death).

3. *Packets-based* techniques, which include duplicate and triplicate packet sending, and packet grouping.

Due to space constraints in this paper, we will only discuss the best performing technique from each of these groups: dead reckoning and smoothing, avatar blinking, and duplicate/triplicate packet sending.

3.1 Dead Reckoning and Smoothing

Dead reckoning (DR) is used to 'guess' a penguin's translation or rotation when the packets holding that information have failed to arrive at the client [4]. We choose to activate DR after one movement packet is lost, and to keep it switched on for at most ten screen updates.

This approach requires packets to be time-stamped, and for a client to estimate how long to wait before a packet is deemed to be lost. The code must also deal with a 'lost' packet turning up after a lengthy delay.

DR is switched on promptly, after only one packet has been lost, so a penguin will keep moving rather than appear unresponsive. DR is switched off after at most ten updates (500 ms in our game), since it becomes very difficult to predict movement accurately after multiple updates.

It is essential to pair DR with smoothing. When a movement packet eventually arrives, smoothing gradually adjusts the penguin's position to relocate and reorientate it to the correct spot. Smoothing is carried out over several screen updates, so a penguin doesn't 'jump' from one position to another.

3.2 Avatar Blinking

Avatar blinking is game-specific: it is triggered when the local player shoots at a penguin, and the client has to wait for the shooting outcome from the remote player. The uncertainty about a penguin's future is denoted by making it blink. This offers immediate feedback to the player, which is more reassuring than have nothing change on screen for perhaps several seconds.

After usability tests, we determined that players find blinking to be helpful for at most a few seconds, after which time it becomes rather irritating. Consequently, a penguin can blink for at most three seconds, which is enough time for a shooting response to arrive when the network is performing at 75% reliability.

3.3 Duplicate/Triplicate Packet Sending

Duplicate/triplicate packet sending makes a client transmit the same packet two or three times to reduce the chance of it being lost en route. One drawback is that the receiver must be able to detect and ignore multiple packet copies. Also, indiscriminate multiple packet sending is a serious consumer of bandwidth. Consequently, we use the technique sparingly, only for important information whose loss would seriously impact the game. Such packets tend to be related to important avatar state changes, such as when a penguin loses life points, or shoots at another penguin. It also helps to correlate the amount of resending to the unreliability of the network.

4. RESULTS

The game was run many times with three clients, and results gathered over several minutes of typical gameplay in each game, and averaged. The tests reported here were carried out with the network set to be 75% reliable.

Three response times measurements were performed: one-way response time for multiple packet actions, one-way response time for single packet actions, and two-way response time.

The mean response times were calculated when no techniques were applied, and again when each of the techniques was switched on individually (i.e. DR and smoothing, avatar blinking, and duplicate/triplicate packets). Finally, all three techniques were switched on together.

The mean response times for the techniques were compared with the mean time when no techniques were enabled, using a standard one-tailed z-test with a 95% level of significance [7]. In the figures below, only the techniques that produced a significant reduction in the mean response time are reported.

4.1 One-way Response Time, Multiple Packet Action

Figure 2 displays mean response times as percentages of the mean response time when no techniques are enabled (shown as the "No Techniques" bar). Consequently, a technique that reduces the time will have a percentage less than 100%. Data for the other response time measures in sections 4.2 and 4.3 are reported in a similar way (see Figures 3 and 4).

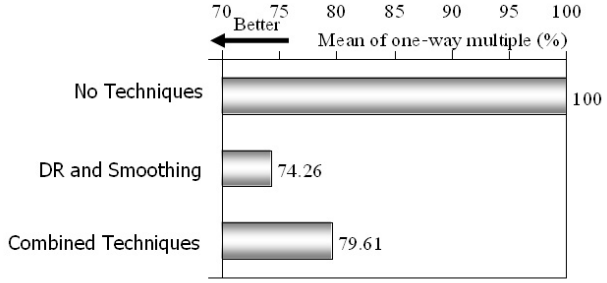

Figure 2. One-way response time, multiple packets.

One-way response times for multiple packet actions are mostly concerned with the processing of avatar movement (translations and rotations). This explains why DR and smoothing reduce the mean response time by a tad over 25% in Figure 2, since that technique compensates for the loss of translation and rotation packets.

Also of interest is that avatar blinking and duplicate/triplicate packets sending (the other two techniques tested here) have no significant effect on this type of responsiveness, and so aren't listed in Figure 2.

4.2 One-way Response Time, Single Packet Action

One-way response times for single packet actions cover the majority of the packets sent in the game, where an action can be codified as a single datagram.

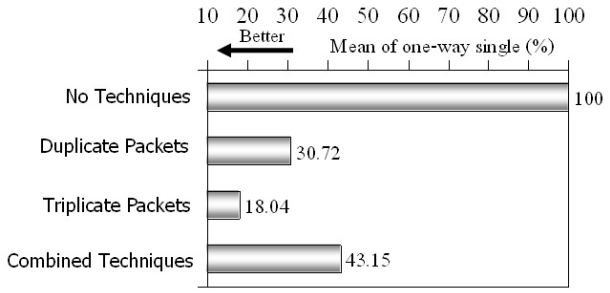

Figure 3. One-way response time, single packets.

Duplicate and triplicate packet sending reduces the response time drastically: by over 80% for triplication which sends the same packet three times (see Figure 3). This reflects the impact that poor network reliability has on game play – at 75% reliability, the "No Techniques" version of the game is almost unplayable.

As the network becomes more reliable (e.g. moving from 75% to 90%), triplicate packet sending becomes slower, and duplicate packets becomes the better performer. The slowdown is caused by the cost of processing and ignoring so many multiple packets.

For this form of response time measurement, DR and smoothing and avatar blinking have no significant effect, so are not shown in Figure 3.

4.3 Two-way Response Time Measurements

In our game, the most important two-way response time measurement is for a player shooting a penguin and waiting for the outcome. Figure 4 shows that avatar blinking is very important for maintaining a good response time, with duplicate/triplicate packet sending also playing a role.

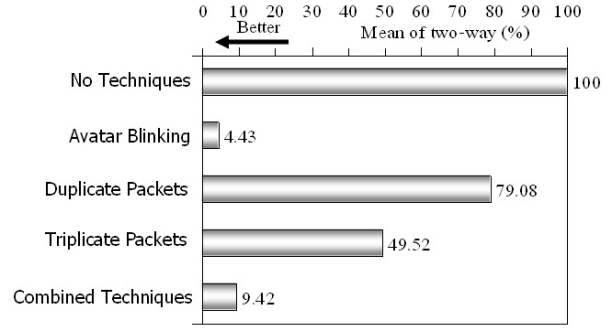

Figure 4: Two-way response time.

Two-way response time is very susceptible to packet loss or delay since it depends on request *and* response packets both being successfully delivered. The loss of one or both of these packets will mean that the associated action cannot be completed.

Avatar blinking does a great job of disguising the delay, which under 75% network reliability conditions may be as much as 2-3 seconds. Duplicate/triplicate packet sending is necessary to ensure that copies of the lost datagrams eventually arrive.

As with the one-way response times for single packet actions in section 4.2, if the network's reliability is increased, then the overhead of triplicate packet sending becomes excessive, and duplicate packet sending becomes the better choice.

5. CONCLUSIONS

Our experiments with a client/server 3D mobile game highlight several issues related to improving client-side response times.

Response time must be measured in multiple ways for a good understanding of how it is affected by varying network reliability and different techniques. One-way response time for *single* packet actions reflects how simple datagram transfer is affected by the network. One-way response time for *multiple* packet actions focuses on more complex data delivery. Two-way response time deals with communication that employs a query/response form.

We have classified the techniques for improving response time into three categories: general, game-specific, and packet-based. A mix of techniques from all these categories gives the best across-the-board improvements. Figures 2, 3, and 4 show that "Combined Techniques" (i.e. dead reckoning and smoothing, avatar blinking, and duplicate/triplicate packet sending) produce mean response times that are 20% to 90% less than the mean response time for the game with no techniques enabled.

Some response time techniques can be politely termed 'tricks', since their aim is to distract the user from the delays inherit in networks with high latency, limited bandwidth, and unreliable packet delivery. Avatar blinking is a good example, but is nevertheless a valuable approach.

6. ACKNOWLEDGMENTS

Our thanks to the Department of Computer Engineering, Faculty of Engineering, at Prince of Songkla University for generously supporting this research.

7. REFERENCES

[1] Fujimoto, R.M. 2000, *Parallel and Distributed Simulation Systems*, John Wiley.

[2] Singhal, S. 1999, *Networked Virtual Environments, Design and Implementation*, Addison-Wesley.

[3] Singhal, S.K. 1996, *Effective Remote Modeling in Large Scale Distributed Simulation and Visualization Environments*, PhD thesis, Dept. of Computer Science, Stanford University.

[4] Pantel, L., Wolf, L.C. 2002, "On the Suitability of Dead Reckoning Schemes for Games", *Proc. of the 1st Workshop on Network and System Support for Games*, 79-84.

[5] Li, S. and Knudsen, J. 2005, *Beginning J2ME: From Novice to Expert*, 3rd Ed., Apress.

[6] Kurose, J.F. and Ross, K.W. 2003, *Computer Networking: A Top-Down Approach Featuring the Internet*, 2nd Ed., Pearson Education.

[7] Mario, F.T. 1998, *Elementary Statistics*, 7th Ed., Addison-Wesley .