Achieving Fairness in Multiplayer Network Games through Automated Latency Balancing

Sebastian Zander
Centre for Advanced Internet Architectures
Swinburne University of Technology
Melbourne, Australia
+61 3 9214 4835
szander@swin.edu.au

Ian Leeder[1]
Centre for Advanced Internet Architectures
Swinburne University of Technology
Melbourne, Australia
+61 3 9214 8089
i_leeder@hotmail.com

Grenville Armitage
Centre for Advanced Internet Architectures
Swinburne University of Technology
Melbourne, Australia
+61 3 9214 8373
garmitage@swin.edu.au

ABSTRACT

Over the past few years, the prominence of multiplayer network gaming has increased dramatically in the Internet. The effect of network delay (lag) on multiplayer network gaming has been studied before. Players with higher delays (whether due to slower connections, congestion or a larger distance to the server) are at a clear disadvantage relative to players with low delay. In this paper we evaluate whether eliminating the delay differences will provide a fairer solution whilst maintaining good gameplay. We have designed and implemented an application that can be used with existing network games to equalize the delay differences. To evaluate the effectiveness of the approach we use a novel method involving computer players (bots) instead of human players. This method provides some advantages over difficult and time-consuming human usability trials. We show that bots experience similar unfairness problems as humans and demonstrate that the application we have developed significantly improves fairness.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed System – *Distributed Applications*; C.4 [**Performance of Systems**]: Measurement Techniques

General Terms

Algorithms, Measurement, Human Factors.

Keywords

Multiplayer Network Games, Fairness, Delay

1. INTRODUCTION

Over the past few years, the prominence of multiplayer network gaming has increased dramatically. There has been a substantial growth in the popularity of network games, growth in the prevalence of game traffic on the Internet [1], and the emergence of network games as an important consideration from a business viewpoint [2]. Computer gaming competitions have become popular and comparable to high-level sporting competitions including prize money, television coverage, and the chance of a title [3].

Fairness is the "quality of treating people equally or in a way that is right or reasonable" [4]. It is a difficult concept to define, especially in terms of game playing. We focus on fairness related to network quality differences between players in terms of network delay, jitter and packet loss. Previous work has shown that latency differences between players can lead to unfairness in fast-paced First Person Shooter (FPS) games (e.g. [5], [6]). The authors of [6] have also found a similar effect for loss but of a much smaller magnitude. To our best knowledge the effect of jitter on multiplayer network games has not yet been sufficiently studied. Therefore in this paper we focus on delay but our proposed approach could be applied in a similar manner to jitter and loss. In this paper we use the term 'fairness' but in game design often the term 'balance' is used instead.

Imagine large-scale international competitions with players from different countries (as illustrated in Figure 1). The players will likely have very different delays to the server. This is just one (admittedly high profile) situation where a fair game server is required. Ideally, every game server should be fair, regardless of each player's location or connection. In fact unfairness caused by delay differences is one of the reasons why most serious competitions still take place in local networks (LANs).

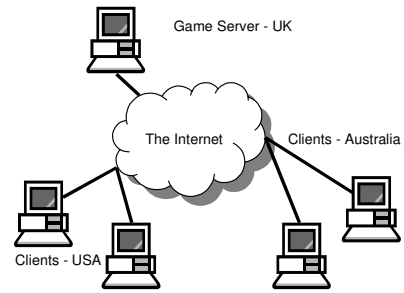

Figure 1: Example of delay-affected games

[1] Ian Leeder worked at CAIA during an R&D student project

Delay is a consequence of many factors – geographical location (propagation delay), access technology (e.g. ADSL, ISDN or dial-up) and transient network conditions (such as congestion). Proper traffic engineering or Quality of Service (QoS) mechanisms can help avoid network congestion and access technologies can be delay optimized (e.g. by enabling 'fast path' for ADSL). However, the propagation delay would still depend on the geographical distance between the communicating parties because information cannot travel faster than the speed of light.

The motivation behind this work is a desire for fair network game servers. Everyone seeks (and demands) a fair chance in any competition, and playing a FPS is exactly that: a game competing for score, respect and acknowledgement of skill. To mitigate the fairness problem in the case of delay differences caused by the network we have developed the Self-Adjusting Game Lagging Utility (SAGLU). SAGLU is a game-independent application that attempts to equalize the delay differences by constantly measuring network delays and adjusting players' total delays by adding artificial lag.

To conclusively demonstrate the effectiveness of our approach it would be necessary to conduct usability trials with human players. However, as discussed in [6] usability trials with human players are resource-intensive and difficult. Great care must be taken when designing such experiments. In this paper we explore a different and novel approach, which we consider as a preliminary alternative to human usability trials. We use client-side computer players (bots) that simulate human players. These bots have a number of limitations and we should not extrapolate too much from their behaviour. However, bots experience similar unfairness effects as human players in case of delay differences and have some advantages: bots are predictable in that they never change their playing style, they have truly equal skills (assuming the same configuration), they are easy to control and can easily perform a large number of experiments (without getting tired). To test the effectiveness of our proposed approach we have carried out several experiments. We show that client-side bots are affected by network delay (differences) similar to human players and demonstrate that the use of SAGLU significantly increases the fairness in games.

The remainder of the paper is structured as follows. Section 2 provides an overview about related work. Section 3 defines our notion of fairness. Section 4 describes the application we have designed and implemented. Section 5 provides the rationale behind the idea of using bots and Section 6 presents the experimental results. Section 7 concludes and outlines future work.

2. RELATED WORK

In [5] the latency tolerance of Quake 3 players was empirically established to be between 150ms and 180ms and it was shown that the average number of kills decreases with increasing latency. Similar studies of the user latency sensitivity for Half-Life show players would not play when latencies are above 225-250ms and that the number of kills significantly decrease with increasing delay ([7], [8]). In [5], [7] and [8] the user sensitivity is inferred by observing the behaviour (e.g. average time on the server, average kill rate) of a large number of users playing FPSs on public servers. While [5] and [7] passively analyse the user behaviour in the face of uncontrolled (normal) network delay, [8] also explores the effect of adding variable levels of artificial delay at the server. In [9] the effects of latency on user performance have been investigated for the Real Time Strategy (RTS) game Warcraft III. The authors find that the performance is not significantly affected by delays ranging from hundreds of milliseconds to several seconds because the nature of RTS emphasizes strategy more than highly interactive aspects.

The use of public game servers limits a researcher's ability to assess player perceived quality in the face of delay and packet loss because the network conditions cannot be exactly controlled and the players cannot be asked about their opinion. Therefore some researchers have conducted usability trials. In [6] two of the present authors investigate the effects of delay and loss on players playing Quake 3 and Halo 1. They also show that different delays can lead to unfairness. Similar work in [10] and [11] investigates the effect of loss and delay on users playing Unreal Tournament 2003. The authors of [12] measured the influence of delay on users playing a simple arcade-style game. The authors of [13] have surveyed players to find out what they think about the Internet.

To the best of our knowledge not many papers exist that deal with fairness in multiplayer computer games. The authors of [14] present a framework for message delivery in real-time multi-player distributed client-server games that attempts to remove the unfair advantage that players with smaller message delays have over players with large message delays. In contrast to [14] we do not define a new framework but rather aim to develop a solution that can improve the fairness for existing FPS games such as Quake or Half-Life.

The implementation of SAGLU is based on the experience gathered with a similar tool we have developed earlier. The Internet Game Lagging Utility (IGLU) [15] was written for the purpose of deterring cheaters on a game server. The idea is that arbitrary delays could be applied, simulating a bad network connection and encouraging cheaters to leave of their own accord instead of having to kick or ban them.

3. FAIRNESS

Fairness refers to all players having equal playing conditions. In this paper we only focus on fairness related to network quality differences between players. In previous work it has been shown that different delays between clients and the server can lead to unfairness (see [5], [6]) giving an edge to players with low delay. Work in [6] also found a similar effect for loss but of a much smaller magnitude. To our best knowledge nobody has yet investigated the influence of jitter. The authors of [16] show that it is difficult to separate between jitter and delay, and work in [17] found that the emulation of jitter is problematic making usability trials difficult. Therefore we focus on delay as the metric of interest. This approach is consistent with other previous work that has identified delay as the most important performance metric for FPS games [18]. However, for a more comprehensive analysis of fairness, other factors such as packet loss rate and jitter should be taken into consideration.

How can fairness be evaluated? In its simplest form, it requires to observe how well two players compete against each other under equal playing conditions, and then compare their performance under different circumstances. Conveniently enough game servers already keep track of a player's performance. The number of kills

(also called frags) per minute can be used as a metric measuring a player's performance. By comparing the performance across different test conditions, fairness can be evaluated.

We define a game as fair if the performance of each player does not depend on network delay differences. We define k_p as the kill rate of player p, which has a delay d_p to the server. Then the mean kill rate of a group of P players with delays d_p approximately equal to d is:

$$\mu(d) = \frac{\sum k_p(d_p)}{P} \quad \text{where } \forall_p \left| d - d_p \right| < \varepsilon \text{ for small } \varepsilon \quad (1)$$

In our experiments we control d_p making sure that we have distinctive groups of player that have similar delays and the delay of different groups differs significantly. A game is fair if there is no statistical significant difference between the mean kill rates of different player groups. We can test this using hypothesis testing. Assuming two player groups with different kill rate means μ_1 and μ_2 the null hypothesis H_0 and the alternative hypothesis H_A are:

$$H_0 : \mu_1 - \mu_2 = 0, H_A : \left| \mu_1 - \mu_2 \right| > 0 \quad (2)$$

If we cannot reject the null hypothesis we can conclude that a game was fair. In the case of more than two player groups pair-wise tests would be required. This fairness definition assumes we can exclude other influencing factors. Our approach of using bots instead of human players helps us to achieve this goal because under equal network conditions all bots should perform equally well when comparing the means of the kill distributions.

4. SAGLU IMPLEMENTATION

The Self-Adjusting Game Lagging Utility (SAGLU) [19] was designed to sit between the game server and the clients, as illustrated in Figure 2. SAGLU has been written in C++ for FreeBSD and is a multithreaded program that uses the pthreads library. It should be noted that an independent machine is not required for SAGLU; it is perfectly capable of running on the same machine as the game server. SAGLU can connect to multiple game servers, polling them periodically to obtain information about current game players and their associated IP address, port and latency (ping).

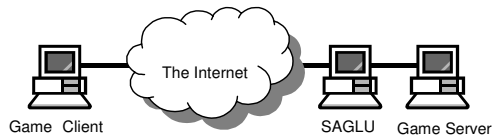

Figure 2: Network configuration of SAGLU

Most multiplayer game servers provide an interface for gathering real-time statistics about the currently running game. Everybody knowing the IP address and port of a running server can access this interface. Depending on the query the information returned may relate to the server state (current map, administrator name, etc.), or to the players on the server (player name, kills, ping). Third-party software such as qstat [20] or gamespy [21] uses this interface. However, this interface does not provide the IP addresses and ports of game clients because that would introduce a massive security problem e.g. players could easily launch Denial of Service (DoS) attacks at rival players. But in order to create the artificial delays SAGLU needs that information for each client.

Most game servers also provide a remote console (rcon) allowing more information to be retrieved, or even commands to be executed on the server. Once a password has been set on the game server, an administrator can send rcon commands to the server either via the game client (connected to the server), or via third-party rcon programs. The rcon interface enables SAGLU to retrieve the IP address and port of each player. This means that SAGLU only be run by the administrator(s) of the game server who have access to the rcon password.

Figure 3 shows the structure of SAGLU. We briefly describe each functional component of SAGLU:

- GameServer: Represents an actual multiplayer game server. It keeps track of the players on that server, and any information, which is specific to the current game being played (GameType).
- GameType: Every specific game that SAGLU knows how to handle is represented as a GameType. Servers are polled in a game-specific manner and the results stored in generic form within SAGLU. This information includes the server name, player names, IP addresses and ping times. Currently SAGLU can query Quake 2, Quake 3, Enemy Territory and Half-Life servers. Other servers can easily be added.
- Player: The information for a player on the particular game server, including the IP address and source port of the client, the current latency between the client and the server etc.
- TrafficShaper: An abstract interface to any software capable of creating artificial delays, specifically the capability to add, edit or delete rules affecting nominated IP packet flows.
- DummyNet: Under FreeBSD SAGLU currently utilises FreeBSD's kernel-resident dummynet traffic shaper [22] to create player specific artificial delay. (Under Linux SAGLU could implement similar functionality e.g. with nistnet [23].)
- Comms: The basic rcon functionality.

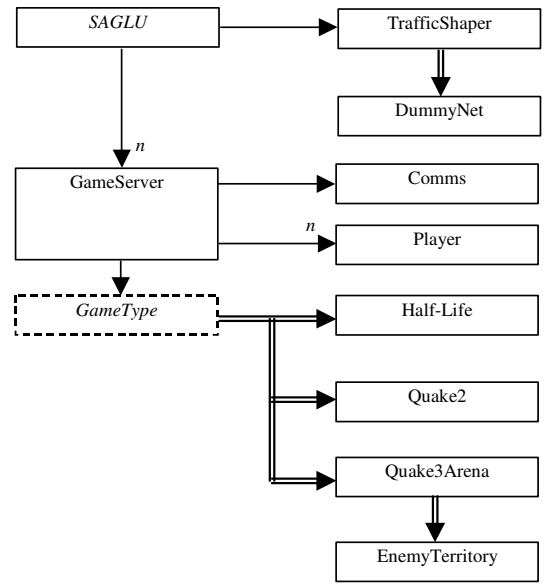

Figure 3: Class diagram of SAGLU

The algorithm for deciding how much delay to add, and how to add it, is the core of SAGLU. This algorithm needs to make a number of decisions:

1. How to determine the additional artificial delay. The additional delay could be based on the highest player delay, a percentage of the highest delay, the average of the highest three delays, etc.
2. How that delay should be added. It could be added immediately or ramped up linearly, exponentially, etc.
3. How often a player's network delay should be measured and the additional delay should be adapted. This is a trade-off between fairness and computational resources available.

It is not a good idea to add large delays to all players simply because one player has a large delay or has dropped from the game and the server is reporting a 999ms delay. As mentioned in Section 2 experimental values have been discovered for the maximum acceptable player delay, which (at the least) can be used as an upper limit for the total delay.

The current implementation of SAGLU uses a rather simple algorithm. In each adaptation cycle SAGLU obtains the network delays of all players from the server and uses the highest delay below the maximum tolerable delay as the target delay. Then it utilizes the traffic shaper to add artificial delay (equal to the difference between the target delay and each player's delay) to all players with less than the target delay. This part of the code is a separate function, so it is a simple task to change the algorithm with little knowledge of the rest of the code.

The adaptation frequency and maximum tolerable latency is configurable separately for each server and game type (because some games are more sensitive to network delay than others). SAGLU also allows the administrator to configure both on a per server basis (over-ruling the game specific settings). Some existing game servers support excluding players from the server that have a higher delay than a configured threshold. This could be used to prevent players from joining a server when their delay is larger than the maximum tolerable delay.

5. USING BOTS

Ideally, a human player's response to SAGLU would be used to determine if an increase in fairness can been achieved. Human players would play under different emulated network conditions and we would collect subjective measures (players' opinion) and objective measures of the players' performance (kills, deaths). Experiments would be done with SAGLU and without SAGLU and we would compare if SAGLU significantly improves the fairness. However, human responses are highly unpredictable and can be influenced by a multitude of unforeseeable factors. Human trials require a careful design and large amounts of tests to ensure statistical reliability.

We use an alternative approach – computer-controlled players (bots) rather than human players. There are two classes of bots: server-side and client-side. The server-side bots are provided standard with most current games, providing adversaries when not enough human players are available. They are built straight into the server, or can be added as patches/mods. These bots do not run over the network and are therefore not influenced by network delay. The second variant of bots, client-side bots, behaves like a real game client. They are usually third-party programs designed to emulate game clients. As such, the game server treats them as real players, and like real players they send real network traffic. The rationale is that client-side bots should be affected by network delay similar to humans.

A problem with bots, especially client-side bots, is that they are far less intelligent than human players. We compensate for this by putting them a simple environment (map) where they can focus on shooting other bots (deathmatch). We avoid large maps that require complex navigation (e.g. lava pits, elevators). The advantage of using bots is that we can eliminate a number of human factors that could easily introduce bias in our studies. Such bias could be avoided at the cost of precise design and large sample size but the effort would be much higher. However, using computer players to evaluate what is in essence a human aspect introduces the questions of how bots react to delay and is their reaction similar to that of human players?

Not only do we need to take into account how delay affects a bot, but also how the bot reacts and/or adjusts its actions. For example, bots are designed to compensate for delay, and predict where the target will be in the future based on the current trajectory and delay estimation. Obviously a human will also do this, but a bot will have much higher precision predicting the future positions of targets than most human players. Additionally a bot may see more than human players. Research has shown that the bots (and this means the game clients too) receive a 360° view of their current position. A normal game client only displays what is 'in front' of the player, but a bot could easily make use of this extra information.

6. EVALUATION

First we describe our experimental setup and then we describe the different experiments and present the results.

6.1 Experimental Setup

In our experiments we use Quake 2 version 3.13 for Linux [24]. Although Quake 2 is an older game and not played anymore we have chosen it because a number of client-side bots exist for Quake 2 [25]. For newer games no client-side bots exist because game designers do not release the protocol specifications (to make cheating more difficult). The release of the protocol specs for Quake 2 has facilitated client-side bot development including research in the area of Artificial Intelligence (AI) (e.g. [26]). Although a number of client-side bots had been developed many have disappeared from the Web and of the bots we were able to download most did not work with our server (the bots either did not connect or crashed shortly after they had connected).

In our experiments we use GoodBot 0.1 [27]. GoodBot only supports line of sight movement, meaning it only moves when opponents are in its line of sight. It does not provide any kind of waypoint navigation, as most current server-side bots do. This means the map must be small and simple. Although it is not required that all bots can see each other all the time, there must be sufficient line of sight between the bots to keep them moving. GoodBot uses prediction including lag compensation when aiming on its targets. When we observed the bot in the game it became apparent that it moves faster than a real game client would enable a player to move. From the source code we found that the bot uses a fixed priority list for the weapons available in the game. Similar to a human player it prefers more powerful weapons.

Unfortunately some very powerful weapons such as the rocket or grenade launcher have a devastating area effect. This is undesirable for our experiments because explosive weapons require much less accuracy, meaning a relatively high kill rate is possible even with large delays.

With the abilities of the bot in mind we choose a suitable map that is simple and does not contain powerful explosive weapons [28]. But the map contains one explosive weapon not on top of the bots priority list (grenades). We use that to test our hypothesis that with increasing delay weapons that require a more precise aim will cause fewer kills.

For the experiments we use a single FreeBSD 2.4GHz PC with 1.25GB of RAM running the Quake 2 server, the client-side bots and SAGLU. A problem we discovered when testing dummynet is that rules using the loopback interface cause twice the configured delay. The reason is that any packets going both to and from the local machine will match the rule twice (see [22]). Therefore in our tests we simply configured dummynet with half the desired delay and adjusted SAGLU accordingly. To achieve a high accuracy for the delay emulation we recompiled the FreeBSD kernel with a tick-timer of 1000Hz (rather than the usual 100Hz).

All our experiments are 15-minute games with 4 bot players. In the beginning of each game the bots join the server with a 1 second delay between each of them. (Initially we tried to join the bots as quick as possible but discovered that this crashes the server.) This does give the first bot an extra three seconds in the game over the last bot, but over a 15-minute trial we assume that to be negligible. There is no difference between the bots except for the order in which they are added and their name. Once the 15-minute game trials were finished, we used kkrcon [29] to retrieve the final scores from the server.

In all experiments we monitored the CPU utilization to make sure there is no delay caused because of insufficient processing time. In all our tests the total CPU utilization was below 70%.

6.2 Experimental Results

6.2.1 How do bots react to delay?

First we tested how the bots react to delay and how their performance decreases with increasing delay. In this experiment we use static symmetric (halve of the total delay in each direction) delays of 100ms, 200ms and 400ms equal for all of the bots. We recorded the kills for all bots playing 15 games.

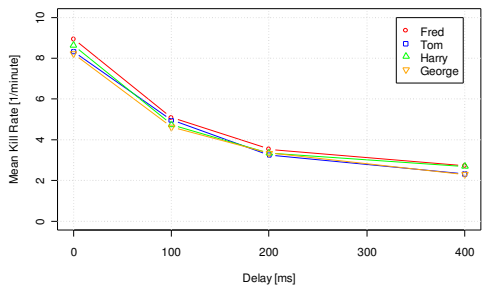

Figure 4: Mean kill rates of the different bots with static delays

Figure 4 shows the mean kill rates of the different bots for increasing delay. Although there are small differences in the kill rate we find them not statistically significant at 99% confidence level. Figure 5 shows the same experiment with dynamic symmetric delays. We use the same mean values of 100ms, 200ms and 400ms but the delay is randomly changed every second using an exponential distribution.

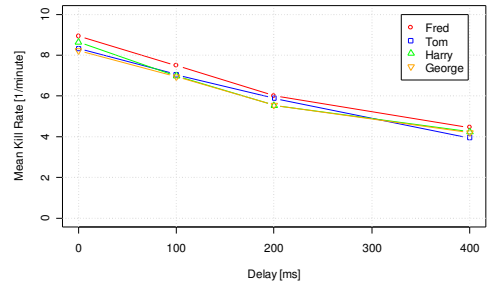

Figure 5: Mean kill rates of the bots with dynamic delays

With exponentially distributed dynamic delays the decrease in kill rate is less severe than for the static delays. Similar to the last figure there are small differences between the different bots but they are not significant at 99% confidence level.

Figure 6 shows the normalized mean number of kills for all bots over increasing static and dynamic delays. The lower and upper ends of the error bars are one standard deviation away from the mean. Because the kill rate not only depends on the players ability but also on the map (size, available weapons) we have normalized the kill rate and compare it with normalized results obtained from [5] (figure 9, average rate of the three best players) and [6] (figure 8, average over both servers). The normalized kill rate is the kill rate fraction players can achieve at certain delays based on their maximum kill rate at zero delay.

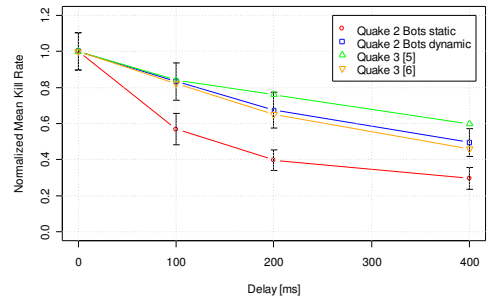

Figure 6: Normalized mean kill rate comparison of bots playing Quake 2 and human players playing Quake 3

The figure shows that in case of dynamic delays the bots perform very similar to human Quake 3 players whereas in case of static delays they perform worse. However, in any case the trend is the same for bots and humans: a constantly decreasing kill rate with increasing delay.

We also recorded what weapons were effectively used to kill the other bot players. Figure 7 shows the percentage of kills caused by each weapon. As delay increases the percentage of kills due to grenades (that have an area affect) is significantly increased as we expected. The shotgun (which also requires less accurate aim because of the spray effect) becomes somewhat more effective than the machinegun at high delays.

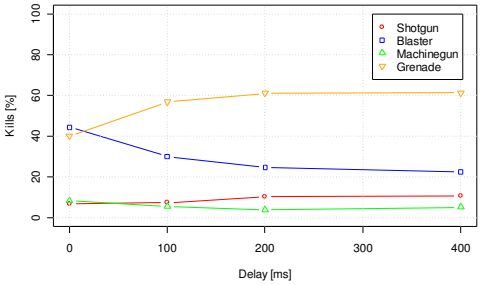

Figure 7: Percentage of kills caused by different weapons

Based on the results we conclude that the different bots on average perform equally well and their performance decreases with increasing latency. As expected, with large delays weapons that have area effects and require less precise aiming caused more kills.

6.2.2 *How do bots react to delay differences?*

Next we tested what happens when different bots experience different delays. Since the bots were all running on the same machine, the UDP traffic had to be separated by port. The port number for each of the bots was obtained directly from the server and used to create the dummynet rules. First we used static symmetric delays where only two randomly chosen bots are delayed while the other two experience no delay and recorded the kills for all bots playing 15 games. Figure 8 shows the mean kill rate of the bots without delay (non-lagged) and bots with delay (lagged) and the standard deviation (error bars).

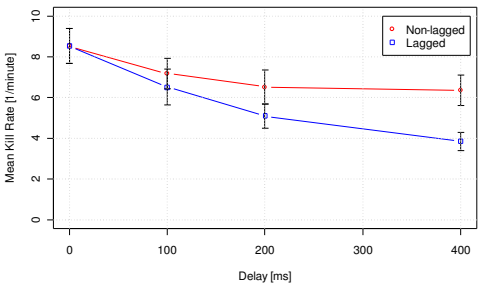

Figure 8: Fairness with static delays

The graph clearly shows that there is a distinct difference in kill rate between the non-lagged bots and the lagged bots. The bots that experience no delay have a clear advantage over the delayed bots. We use t-tests to check if the differences are statistically significant. We find that for 100ms, 200ms and 400ms the differences in the kill rate are statistically significant at 99% confidence level (p-values: 0.002, 5.0e-10, 2.2e-16).

Figure 9 shows the same experiment with dynamic delays (changing once per second) but the same mean values (see previous section). The difference between non-lagged and lagged bots is smaller as for the static delays but the t-tests show it is still significant for 100ms, 200ms and 400ms at 99% confidence level (p-values: 0.01, 0.002, 3.6e-10).

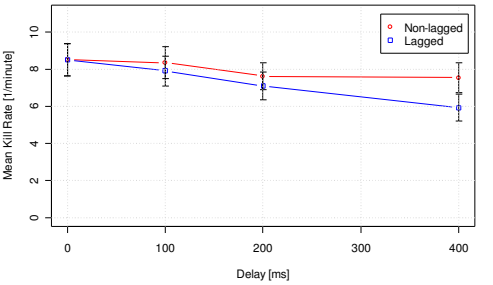

Figure 9: Fairness with dynamic delays

Our results show that the bots experience delay differences similar to human players in that the bots with lower delay have a clear advantage over bots with larger delay. However, the difference seems to be smaller than what was previously observed for human players. We believe this is because the bot's prediction algorithm and lag compensation is better than that of most humans.

6.2.3 *Can SAGLU achieve fairness?*

We repeated both tests described in the previous section with SAGLU enabled. We configured SAGLU with an adaptation interval of 5 seconds and a maximum tolerable delay of 600ms. Figure 10 shows the mean kill rates and standard deviations for both players groups and static delays.

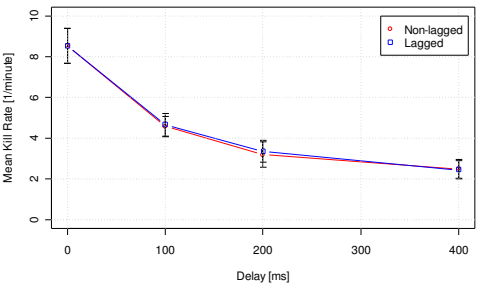

Figure 10: Fairness with SAGLU and static delays

The t-tests indicate that there is no significant difference in the kill rates for 100ms, 200ms and 400ms. Figure 11 shows the mean kill rates and standard deviations for both player groups and dynamic delays (the maximum network delay was always lower than SAGLU's maximum delay tolerance).

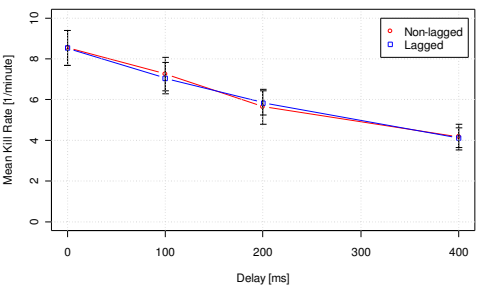

Figure 11: Fairness with SAGLU and dynamic delays

With SAGLU enabled we cannot find a statistical significant difference between the two groups for 100ms, 200ms and 400ms.

The results obtained from these experiments show that SAGLU has evened the playing field considerably even when adapting at a lower rate (every five seconds) than our synthetic network delay changes (once per second). SAGLU ensures players with a low delay lose their advantage over players with high delay. Evening the delays causes a decrease in kill rate. While one would expect the mean kill rate of the players with zero network delay (only artificial delay applied by SAGLU) to drop, in our experiments both mean kill rates decreased to a value below the mean kill rate players with large network delay had in the unfair games. This behaviour seems to be specific to our experimental setup because it cannot be confirmed by previous work. However, a similar effect occurs in professional sport competitions e.g. in the Formula-1 there are many regulations that cause all the cars to be slower than they could be to increase fairness and make the competition more interesting.

The maximum delay of 400ms we used in our experiments is obviously larger than what most human players would usually tolerate. We chose such a high maximum delay to make sure we observed a clear reaction from the bots. As existing work shows the maximum delay tolerance of humans is somewhere between 150ms and 250ms depending on the specific game. Therefore our results for 100ms and 200ms are clearly relevant. We did not use small delay values in our tests because related work in [5] and [6] as well as preliminary tests with GoodBot found no significant performance decrease for delays around 50ms. However, some very good (professional) players claim that such small delays already affect their performance. They could benefit from the use of SAGLU at such low delays. Even if small delay differences have no measurable impact on fairness, running SAGLU could achieve a psychological effect in the players' minds making them believe that with SAGLU the game is fair.

7. CONCLUSIONS AND FUTURE WORK

Previous work has shown that unfairness caused by network delay differences between players is a problem for past-paced multiplayer network games. We have designed and implemented the Self-Adjusting Game Lagging Utility (SAGLU), an application that can be used with existing games to equalize the delay differences and make the games fair. To evaluate the effectiveness of our approach we have used a novel method utilizing computer players (bots). We have shown that the bots react similar to delay and experience similar unfairness problems as humans. We also demonstrated that SAGLU significantly improves the fairness of games.

We plan to do more tests with different maps and bot configurations to further verify the effectiveness of SAGLU, refine the algorithm and fine-tune the parameters. Ultimately we plan to run usability trials with human players because only the reaction of human players would allow us to properly dimension all the parameters such as the maximum tolerable delay and adaptation frequency. Measuring the player latencies by polling the information from the game server introduces additional CPU and network load on the game server. We plan to characterize this additional load and optimize the adaptation interval so that fairness is achieved while the effects on the game server performance are minimized. In case SAGLU is used for multiple game servers it should be ensured that the polling of different servers is not synchronous, as this would create traffic bursts in the network that could cause short-term congestion. If one SAGLU box is used for a very large number of game servers the performance of dummynet with hundreds or thousands of rules should to be evaluated.

8. ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their useful comments that helped improving the final version of this paper.

9. REFERENCES

[1] S. McCreary and k. claffy. *Trends in wide area IP traffic patterns - A view from Ames Internet Exchange*. in ITC Specialist Seminar, Monterey, CA, 18-20 Sep 2000.

[2] ISP planet news, http://www.isp-planet.com/news/2002/gamez_021202.html, December 2002 (as of April 2005).

[3] Marcus Graham. *The Rise of Pro Gaming*. Game Daily, http://biz.gamedaily.com/features.asp?article_id=8858&section=myturn, February 2005 (as of April 2005).

[4] Cambridge dictionary: http://dictionary.cambridge.org/ (as of April 2005).

[5] G.Armitage. *An Experimental Estimation of Latency Sensitivity in Multiplayer Quake3*. Proceedings 11th IEEE International Conference on Networks (ICON) 2003, Sydney, Australia, September 2003.

[6] S. Zander, G. Armitage. *Empirically Measuring the QoS Sensitivity of Interactive Online Game Players*. ATNAC 2004, Sydney, Australia, December 2004.

[7] T. Henderson. *Latency and user behaviour on a multiplayer game server*. Proceedings of the 3rd International Workshop on Networked Group Communications (NGC), London, UK, November 2001.

[8] T. Henderson, S. Bhati. *Networked games — a QoS-sensitive application for QoS-insensitive users?*. SIGCOMM RIPQoS Workshop 2003, Karlsruhe, Germany, August 2003.

[9] N. Sheldon, E. Girard, S. Borg, M. Claypool, E. Agu. *The Effect of Latency on User Performance in Warcraft III*. In Proceedings of ACM Network and System Support for Games (NetGames), Redwood City, California, USA, May 2003.

[10] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, M. Claypool. *The Effects of Loss and Latency on User Performance in Unreal Tournament 2003*. NetGames2004 Workshop, SIGCOMM2004, Portland, Oregon, USA, August 2004.

[11] P. Quax, P. Monsieurs, W. Lamotte, D. De Vleeschauwer, N. Degrande. *Objective and Subjective Evaluation of the Influence of Small Amounts of Delay and Jitter on a Recent First Person Shooter Game*. NetGames2004 Workshop, SIGCOMM2004, Portland, Oregon, USA, August 2004.

[12] C. Schaefer, T. Enderes, H. Ritter, M. Zitterbart, "Subjective Quality Assessment for Multiplayer Real-Time Games", Proceedings of the first ACM workshop on Network and system support for games (NetGames2002), April 2002.

[13] Manuel Oliveira, Tristan Henderson. *What Do Online Gamers Really Think of the Internet?*. Proceedings of the 2nd workshop on Network and system support for games NetGames NetGames 2003, Redwood City, CA, May 2003.

[14] Katherine Guo and Sarit Mukherjee and Sampath Rangarajan and Sanjoy Paul. *A fair message exchange framework for distributed multi-player games*. Proceedings of the 2nd workshop on Network and system support for games NetGames 2003, Redwood City, California, USA, 2003.

[15] IGLU, http://caia.swin.edu.au/genius/genius-tools.html (as of April 2005)

[16] G.Armitage, L.Stewart. *Limitations of using Real-World, Public Servers to Estimate Jitter Tolerance Of First Person Shooter Games*. ACM SIGCHI ACE2004 conference, Singapore, June 2004.

[17] G.Armitage, L.Stewart. *Some Thoughts on Emulating Jitter for User Experience Trials*. Proceedings of the NetGames 2004 Workshop, ACM SIGCOMM2004, Portland, Oregon, USA, August 2004.

[18] J. Faerber. *Network game traffic modelling*. Proceedings of the 1st ACM workshop on Network and System Support for games, April 2002.

[19] SAGLU, http://caia.swin.edu.au/genius/tools/saglu-0.1.tar.gz (as of April 2005)

[20] qstat, http://www.qstat.org/ (as of April 2005)

[21] gamespy, http://www.gamespy.com/ (as of April 2005)

[22] Luigi Rizzo, dummynet, http://info.iet.unipi.it/~luigi/ip_dummynet/ (as of April 2005)

[23] nistnet: http://www-x.antd.nist.gov/nistnet/ (as of April 2005)

[24] Quake2, http://www.idsoftware.com/games/quake/quake2/ (as of April 2004)

[25] Quake2 client-side bots, http://www.planetquake.com/mikeBot/remote-resources.html (as of April 2005)

[26] John Laird's Artificial Intelligence & Computer Games Research, http://ai.eecs.umich.edu/people/laird/gamesresearch.html (as of April 2005)

[27] GoodBot 0.1, Jens Vaasjo, jvaasjo@iname.com, http://www.gamers.org/pub/idgames2/quake2/dlls/bots/ (as of April 2005)

[28] jarduel3b.bsp, http://www.jaruzel.com/quake2.shtml (as of AprilOctober 20054)

[29] kkrcon, http://kkrcon.sourceforge.net/ (as of AprilFebruary 2005)