

Fairness in Multi-Player Online Games on Deadline-Based Networks

Anh Le

Yanni Ellen Liu

Department of Computer Science, University of Manitoba, Winnipeg, MB R3T 2N2, Canada

Abstract—The fairness issue in multi-player online games (MOGs) on deadline-based networks is investigated. A particular type of MOGs, namely the first-person-shooter (FPS) game, is examined. Potential unfair scenarios in FPS games due to disparate network delays are analyzed. A fairness definition and two fairness metrics are introduced. Two strategies to achieve fairness in FPS games on a deadline-based network are developed. The first strategy is based on end-system delay compensation; the other is based on the characteristics of the underlying network. The performance of the developed strategies is evaluated by simulation. Experiments show that by employing these two strategies, game fairness can be significantly improved.

I. INTRODUCTION

Recently, the number of real-time applications running on packet-switched computer networks increased rapidly. They vary from stock quote update and online auction, to audio, video conferencing, Internet telephony, and real-time interactive online games. These applications require timely delivery of real-time data across the transport network. In order to support this, quality of service (QoS) support at the transport network is required.

Deadline-based network resource management [1] is a new approach to supporting real-time applications in packet-switched networks. In this approach, a pair (size, deadline) is specified for each application data unit (ADU). An ADU can be a file, a state update message in an online game, or an audio or video frame in multimedia applications. The ADU deadline specifies the absolute time at which the ADU should be received by the receiving application. For delivery, an ADU may be fragmented and sent in more than one packet. The ADU deadline is mapped to packet deadlines, which are carried by packets and used by routers for channel scheduling. Deadline-based scheduling is employed in routers. When compared to the first-come first-served scheduling, which is used on the classic Internet, deadline-based channel scheduling was shown to yield better performance in terms of the percentage of ADUs that are delivered on time [1]. In the literature, the effectiveness of deadline-based scheduling has been evaluated assuming generic real-time traffic rather than traffic from more specific types of real-time applications. In this research, as a step towards real-world application support, we investigate the capability of deadline-based networks in support of MOGs.

MOGs are computer games in which multiple game players simultaneously participate in a game session over a computer network. MOGs are increasingly popular on today's Internet. The implementation of MOGs often adopts a client-server

architecture in which each game client is connected to a game server via a computer network. State update messages are transmitted between the game clients and the game server. There are various types of MOGs, among which the FPS games often have the most stringent requirement on the delay performance of the underlying network because of the highly interactive nature of such games. Because of its high QoS demand on the network, this research focuses on the support to FPS games on deadline-based networks.

One of the most important requirements of a FPS game is fairness. In common terms, fairness gives each player an equal opportunity to win the game. More specifically, fairness may be achieved if (i) the messages sent from the server can be received by all the clients after the same amount of time, and (ii) the messages sent from all clients take the same amount of time to be received by the server. Our goal is to come up with strategies providing fair treatment among game clients on a deadline-based network.

On a packet-switched network, the packet end-to-end delay includes transmission delay, propagation delay, processing delay, and queuing delay. When routing is fixed, so are the first two delays. The processing delay is determined by hardware/software overhead within the end systems as well as the routers, and is outside the scope of this study. The queuing delay is affected by network resource management strategies such as channel scheduling. With deadline-based channel scheduling, packets that carry more urgent deadlines are given higher priority than packets that carry less urgent deadlines [1]. Thus by changing the deadline of a packet, one could give it higher priority in the network. This may lead to reduced queuing delay, thus reduced end-to-end delay. We term fairness strategies based on network resource management *network-based* strategies. Another fairness strategy investigated is *delay compensation*. Within a tolerable delay bound which is often needed to maintain realistic interactivity, disparate delays to and from different game players are compensated at the server and client sides, so that the average one-way delays among the players are about equal.

We first analyze unfair scenarios in FPS games due to disparate network delays and introduce a fairness definition and two fairness metrics. We then present our network-based and delay compensation based fairness strategies. Simulation results show that these strategies can significantly improve the fairness performance of FPS games on a deadline-based network. The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 analyzes unfair scenarios

and introduces the fairness definition and metrics. Section 4 presents our proposed strategies. Sections 5 and 6 evaluate their performance. Finally, Section 7 concludes this study.

II. RELATED WORK

Fairness in MOGs has been studied in the literature. In [2], a strategy was proposed to improve fairness by dropping obsolete events, thus speeding up the delivery of game events. The end-to-end latency for each event among all the clients is kept below a game interactivity threshold (GIT). Then extra delay is added for each event so that all latencies equal GIT. Our strategies adopt GIT; however, extra delays are only added to a subset of packets. Moreover, we also look into the fairness issue of client-to-server packets, while [2] only discussed the fairness issue of server-to-client packets.

In [3], a framework was devised to guarantee that the order of messages at the server is the same as the order in which they are originally sent from the clients. This framework requires the usage of added proxy servers at both client and server sides. The clients' proxy servers track the user response times of the clients to an event generated by the server, and include the response times in the packets that are sent from the clients to the server. The server-side proxy sorts the packets in increasing order of the clients' response times. Shorter response time means faster reaction to the event; this in turn means the packet is generated before other packets. The server-side proxy then sends these packets to the server to process them in that order. The trade-off is the overhead of running proxies and sorting packets. Our proposed strategies take a different approach, we look at the end-to-end network delays which are at a protocol layer below the scheme in [3]. Hence, the framework from [3] could be used in addition to our strategies.

III. FAIRNESS IN FPS GAMES

In this section we present two unfair scenarios, give the definitions of fairness in FPS games, and define fairness metrics. For implementation, we also discuss the time synchronization issue.

A. Unfair Treatment in FPS Games

When one-way delays between the server and the clients are different, unfairness might arise. We illustrate this with two scenarios (see Fig. 1). In Scenario I, Client 1 opens fire and kills object A before Client 2 does. We assume that state update packets are sent right after the events (The case when packets are not sent right after the events is discussed in detail in [13]). Since there is a difference between the one-way delays from the two clients to the server, the state update message from Client 2 arrives at the server first, and the server credits Client 2 for the kill instead of Client 1. In Scenario II, a game server sends out two "encounter messages" to Clients 1 and 2 respectively, indicating that they become visible to each other in a game scene. The two messages are sent back-to-back at the same time. However, due to the difference in one-way delays, Client 2 receives the packet earlier than Client

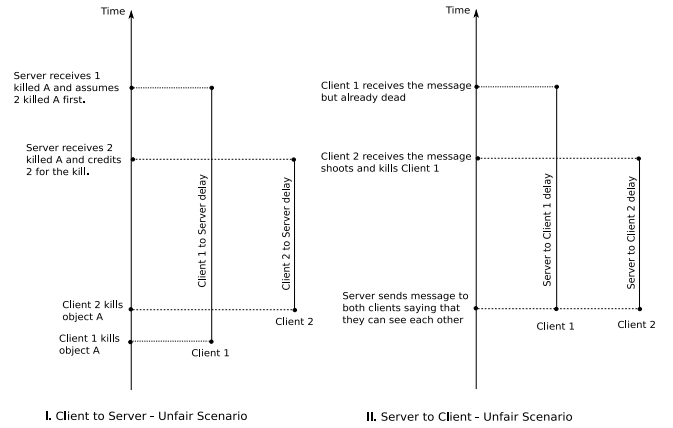


Fig. 1. Unfair scenarios

1. Therefore Client 2 can see Client 1 earlier, and can open fire and kill Client 1 even before Client 1 receives the encounter message. This is clearly unfair to Client 1. In both scenarios, the unfairness can be avoided if packets from both clients have the same one-way delay to the server.

B. Definition of Fairness

For a FPS game, a game session is said to be *absolutely fair* if the packets sent from all the clients to the server have the same one-way delay and all the packets sent from the server to the clients have the same one-way delay.

For a FPS game, a game session is said to be *fair* if the average one-way delays of the packets sent from each client to the server are the same, and the average one-way delays of the packets sent from the server to each client are the same.

In a packet-switched network, it is very expensive to achieve absolute fairness; the implementation would incur the emulation of a circuit similar to one in circuit-switched networks. In contrast, it is more feasible to achieve fairness. As FPS games are often considered "soft" real-time, ensuring fairness may be adequate. We aim at achieving fairness in this study.

C. Fairness Metrics

In order to achieve fairness in a FPS game session, a fairness measure is called upon. We introduce two fairness metrics. There are two types of packets in a game session: packets sent from the clients to the server, and packets sent from the server to the clients. Both of them contribute to the fairness of the game session. We examine them in sequence.

1) *Client-to-Server Traffic*. Let $(d_i)_k$ ($i = 1 \dots n$) be the end-to-end delay of packet k from client i to the server, where n is the number of clients in the game session. Let θ_i ($i = 1 \dots n$) be the *average* end-to-end delay of all the packets sent from client i to the server. If there are m_i packets sent from client i to the server then: $\theta_i = \frac{\sum_{k=1}^{m_i} (d_i)_k}{m_i}$. Let ϵ be the average end-to-end delay of all packets sent from *all the clients* to the server. Hence, $\epsilon = \frac{\sum_{i=1}^n \theta_i}{n}$. Also, let δ_i ($i = 1 \dots n$) be the absolute value of the difference between θ_i and ϵ : $\delta_i = |\theta_i - \epsilon|$. The larger the δ_i is, the further the average delay of

client i deviates from the total average. In this case, the game is considered less fair with respect to client i . The first fairness metric we define is called *fairness index 1*, denoted by \mathbf{F}_1 . \mathbf{F}_1 is calculated by: $\mathbf{F}_1 = \sum_{i=1}^n \delta_i$. Note that the smaller the value of \mathbf{F}_1 , the fairer the game session. If all the clients have the same end-to-end delays then \mathbf{F}_1 is zero. On the other hand, if there are differences among the θ_i 's then \mathbf{F}_1 is positive, and the larger the differences are, the larger \mathbf{F}_1 becomes.

2) *Server-to-Client Traffic*. Each type of game has a *game interactivity threshold* (GIT). Introduced in [2], GIT specifies the minimum end-to-end delay required for the server-to-client packets to achieve interactivity in a FPS game. If the end-to-end delay of a server-to-client packet is larger than GIT then the player at the client receiving that packet would feel loss of interactivity (i.e. the game events are not displayed smoothly). This leads to an unfair game session. The same idea used for the first type of packets is applied here. However, GIT is used in place of the overall average delay ϵ . We define another fairness metric, *fairness index 2* \mathbf{F}_2 , which is measured at regular time intervals. \mathbf{F}_2 is calculated by: $\mathbf{F}_2 = \sum_{i=1}^n |\theta_i^* - \text{GIT}|$. θ_i^* ($i = 1 \dots n$) is the *average end-to-end delay* of all the packets sent from the server to client i , where n is the number of clients. In a degenerate case, when all the θ^* 's are less than the default GIT, the GIT value for the next interval is updated to be the largest θ^* . Otherwise, the next interval have the default GIT value. Note that if all the packets have GIT as the end-to-end delay then \mathbf{F}_2 equals zero and the fairness is also achieved. However, if there are packets which have end-to-end delay θ^* 's different from GIT then \mathbf{F}_2 is positive. Moreover, the smaller the δ^* 's are the smaller \mathbf{F}_2 gets. Smaller δ^* 's indicate more uniform gaming experiences for all the players. Consequently, smaller \mathbf{F}_2 indicates a fairer game session.

D. Clock Synchronization

By definition, obtaining both \mathbf{F}_1 and \mathbf{F}_2 requires the measurement of one-way delays. One-way delays can be measured using timestamps. The times a packet is sent by the sender and received by the receiver are read from sender's and receiver's clocks respectively. Assuming clock synchronization between the two clocks, the difference between the two readings would be the one-way delay. However, clocks in a network environment can not be perfectly synchronized. In this case, \mathbf{F}_1 (or \mathbf{F}_2) is only meaningful if the differences between the two clocks are negligible comparing to δ_i (or δ_i^* , which is defined by $|\theta_i^* - \text{GIT}|$). One way to achieve high accuracy clock synchronization is using the Global Positioning System (GPS). GPS can produce up to 20 ns clock accuracy with respect to UTC [4], which is the Coordinated Universal Time. Adding hardware errors brings the accuracy to about 5 μ s. This level of accuracy is sufficient for the proposed strategies since the delays are measured in ms. One can also achieve high accuracy clock synchronization without equipping a GPS receiver at every node. As discussed in [5], Cesiumspray is a clock service that can give high accuracy clock synchronization by deploying one GPS receiver per group of nodes, all the nodes

in a group are on a local area network. Cesiumspray can render up to 1 ms accuracy and could be adequate if one-way delays have an order of magnitude differences.

IV. ACHIEVING FAIRNESS

As discussed in the previous section, fairness can be achieved by minimizing both \mathbf{F}_1 and \mathbf{F}_2 . This section discusses in detail how to achieve fairness in a FPS game session. We first introduce some terminologies related to deadline-based networks: 1) Packet deadline: The absolute time at which a packet should arrive at the network layer of the receiver. 2) End-to-end latency of a packet: the sum of total transmission delays and propagation delays of a packet along the packet path. It is fixed once the routing is fixed. 3) End-to-end deadline of a packet: the difference between the packet arrival time at the network at the sender, and the packet deadline. In other words, it is the allowable delay for a packet to get from the source to the destination and must at least equal the end-to-end latency. Note that on the current Internet, deadline does not exist or can be considered infinite.

A. Minimizing \mathbf{F}_1

The first step to achieve fairness is minimizing \mathbf{F}_1 . The measurement of \mathbf{F}_1 is first presented, the minimization method is described afterwards.

Measuring \mathbf{F}_1 : We measure \mathbf{F}_1 periodically. \mathbf{F}_1 can be calculated either over an interval of time or over a certain number of packets. There is no firm requirement for the length of the period. As a start, in this study, \mathbf{F}_1 is calculated *periodically every 5 s*.

Minimizing \mathbf{F}_1 : Base on the \mathbf{F}_1 calculated for the time interval \mathbf{I}_j , actions can be taken to lower \mathbf{F}_1 for the interval \mathbf{I}_{j+1} . This study introduces two strategies to reduce \mathbf{F}_1 : delay compensation strategy and network-based strategy.

1) In the "delay compensation" strategy, if there are clients who have the average delay θ 's lower than the total average delay ϵ in the interval \mathbf{I}_j , then extra delays will be added to the packets sent from these clients in the interval \mathbf{I}_{j+1} , before they are considered by the game server. The amount of delay added to the packets sent from client i is $\epsilon - \theta_i$. Hence, the expected θ_i for those clients in the interval \mathbf{I}_{j+1} is ϵ .

In the interval \mathbf{I}_{j+1} , for each client, both the incremented and the non-incremented θ 's will be calculated. The total average delay ϵ for the interval \mathbf{I}_{j+1} is calculated based on the incremented θ 's. After that, based on the value of the new ϵ and the non-incremented θ 's in interval \mathbf{I}_{j+1} , the amount of delays added are adjusted accordingly for the interval \mathbf{I}_{j+2} .

All the calculations are done by the server. The average delay θ_i can be updated on the fly whenever a packet from client i arrives at the server. At the end of each interval, ϵ is calculated to determine the amount of delay that must be added in the next interval.

This is an iterative process and the goal is to increase the delays of the packets from those clients that have shorter one-way delays to the game server than others (e.g., the clients that

are closer to the server). Consequently, these clients would not gain an advantages over others.

2) In the “network-based” strategy, if there are clients who have the average delay θ 's larger than the total average delay ϵ in the interval \mathbf{I}_j , then the deadlines of the packets sent from these clients in the interval \mathbf{I}_{j+1} are adjusted downwards, so that the end-to-end delays of these clients' packets may be reduced. This is because more urgent deadlines may give them higher priorities at each hop on a deadline-based network and may lead to reduced queuing delay. This way, the θ 's can be lowered.

Because GIT of 100 ms is suggested for FPS games in the literature [2], the end-to-end deadline of packets used in this study is 100 ms. Given this, the packet deadline carried in each packet can be obtained by adding 100 ms to the packet arrival time at the network. If the deadlines of the packets sent from a client to the server needed to be adjusted in the interval \mathbf{I}_{j+1} , at the end of the interval \mathbf{I}_j , the server sends a control packet to the client to inform the adjustment amount. When the client receives the control packet from the server, it compares the current end-to-end deadline to the end-to-end latency. If the difference of the end-to-end deadline and the end-to-end latency is not less than a *decrement amount*, it reduces the deadlines of the packets sent in the interval \mathbf{I}_{j+1} by the decrement amount. This comparison is needed because the end-to-end deadline must be no less than the end-to-end latency to be reasonable. If there is no room to reduce the deadline, the client ignores the control packet and does not adjust the packet deadline. This also means that at this time there is no more improvement of fairness that can be achieved by this strategy.

The goal of this process is to reduce the packet delays of the clients who have larger one-way delays to the game server than others (e.g., the clients further away from the server). Consequently, these clients would not have an disadvantages over others.

B. Minimizing F_2

The second step to achieve fairness is to reduce F_2 .

Measuring F_2 : F_2 is also measured periodically. We also use 5 s intervals. Let \mathbf{I} be a 5 s interval. For each interval \mathbf{I}_j , the end-to-end delays (d_i^*)'s of the packets sent from the server to the client i ($i = 1 \dots n$) are measured. These delays are used to calculate θ^* 's and $(F_2)_j$ for the interval \mathbf{I}_j .

Minimizing F_2 : Base on the F_2 calculated for the interval \mathbf{I}_j , actions can be taken to lower F_2 for the interval \mathbf{I}_{j+1} . There are also two strategies to minimize F_2 . Both approaches are similar to the ones used to minimize F_1 . However, GIT is used in place of ϵ .

V. PERFORMANCE MODEL

We evaluate the performance of our proposed strategy using simulation. This section describes the performance model which is built to evaluate the effectiveness of the proposed strategies. Next section presents the simulation results. The performance model that we used is a discrete event simulator

that is developed in Java. It simulates the packet dynamics in a deadline-based network at the network layer.

A. Network Model

The network model consists of 11 nodes. It is a scaled down version of the Abilene backbone network [10]. In this network, each node denotes a city in the US. The distances between nodes are preserved. However, to speed up the simulation, the link bandwidth is scaled down by a factor of 10. The propagation delays are calculated from the distances with a propagation speed of 120 miles per ms. Figure 2 shows the

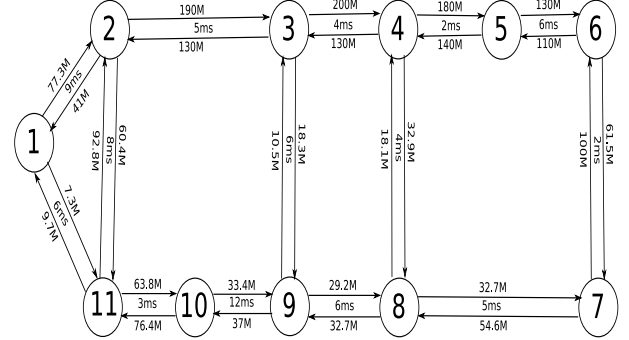


Fig. 2. Network Model

network topology as well as the bandwidth and the propagation delay of each link. In this study, we assume that routing is fixed. The maximum transmission unit (MTU) is 1400 bytes. It determines the maximum packet size. An ADU larger than MTU is subjected to fragmentation at the sender. In FPS games, the size of state update messages are normally small, thus each state update message can be accommodated within just one packet, no ADU fragmentation and re-assembly is needed for the game packets. For simplicity, at each outgoing channel of each router, an infinite buffer size is assumed.

B. Traffic Model

Two types of traffic are simulated in this network. One is a benchmark game session whose fairness performance is measured and compared between with and without the proposed fairness strategies. The traffic generated by this game session is called “foreground traffic”. The other type of traffic is the “background traffic” that shares the same network as the foreground game session. These two types of traffic are described below.

1) *Background Traffic:* The background traffic consists of three components: constant bit rate music streams, variable bit rate movie streams, and a FPS game session.

There are multiple 128 kbps music streams sent from each node to every other node. Each music packet has size 400 bytes. Music packet inter-arrival time is 0.04 sec, i.e, one packet every 25 ms. The music packet size is chosen to accommodate a standard 44.1 kHz MP3 sound frame which is about 400 bytes [11]. One movie stream is sent from each node to every other node. The movie is Die Hard 3 and is encoded using MPEG 4 with medium quality. The movies frames are

read from a trace file. The maximum frame size is 65 kbit, the minimum frame size is 0.2 kbit. The trace file is obtained from [12].

The *game session* includes one server at node 3 serving a FPS game and 10 players at 10 other nodes. Game traffic models are available in the literature [6], [7], [8], [9]. As proof of concept, we adopt the model in [6]. If n , the number of clients, is 10, then: (i) from a client to the server: packet sizes follow a log normal distribution with mean 75 bytes and standard deviation 3 bytes. Packet inter-arrival time is 11 ms. (ii) from the server to a client: packet sizes follow a log normal distribution with mean $= 13 \cdot n + 65 = 195$ bytes and standard deviation $= 2 \cdot n + 7 = 27$ bytes. The packet inter-arrival time is fixed at 50 ms. Initially, background music, movie, and game traffic packets have end-to-end deadlines of 200 ms. The control packet is assumed to be 50 bytes long and to have end-to-end deadline of 20 ms.

2) *Benchmark Game Session*: This game session is considered foreground traffic and is the subject of the study. The game session is the same as the game session used for the background traffic. However, the server is located at node 4. The foreground game traffic packet has end-to-end deadline of 100 ms.

VI. SIMULATION RESULTS

The performance measures of interest of this study include both F_1 and F_2 . Each experiment is repeated ten times. For each experiment, the sample mean and the 95% confidence intervals are computed. Because the width of the confidence interval is always lower than 4% of the sample mean, we omit confidence intervals from our results.

In our experiments, for both F_1 and F_2 , we examine how disparate network load levels and different traffic deadline urgency levels affect the effectiveness of the proposed strategies.

A. F_1 Related Results

1) Effect of Network Load

Three different network load levels are experimented. They correspond to three different utilization levels on the bottleneck channel, through which the foreground client-to-server game traffic passes. The three utilization levels are 95%, 90%, and 85%. These load levels are implemented by varying the number of background music streams that pass through the same bottleneck. In Fig. 3, the values of F_1 for the game traffic that passes through the bottleneck are plotted for four scenarios: (i) without any fairness strategies, (ii) with the “delay compensation” strategy in which extra delays are added to the packets that have one-way delays lower than the average one-way delay of all the packets, (iii) with the “network-based” strategy in which the deadlines of the packets that have one-way delay larger than the average delay of all the packets are adjusted, and (iv) when with both strategies. We first observe from Fig. 3 that the value of F_1 when “network-based” strategy is used is lower than when without any strategies. The value of F_1 is near zero when “delay-compensation” strategy is used, either alone or together with

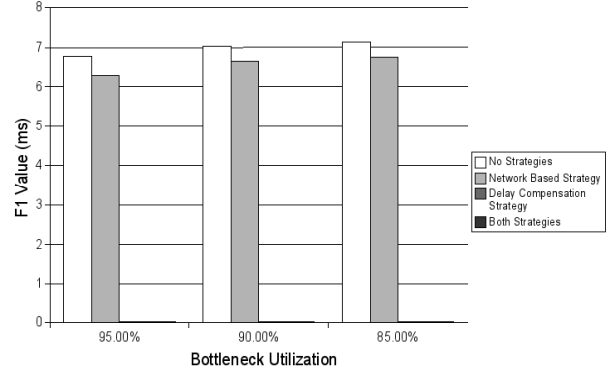


Fig. 3. Effect of Network Load on F_1

“network-based” strategy. These demonstrate the effectiveness of the proposed strategies. Secondly, across three different load levels, the improvement given from “network-based” strategy is slightly larger as the load level increases. The value of F_1 is reduced by 7.24%, 5.68%, and 5.27% at 95%, 90%, and 85% utilizations respectively. The “delay-compensation” strategy is able to reduce the value of F_1 to zero in all load levels. We conclude that the “delay-compensation” strategy works better than the “network-based” strategy; regardless of the load, both strategies lead to significant fairness improvement.

2) Effect of Deadline Urgency

In this experiment, we examine the effect of different deadline urgency levels on the performance of the two proposed strategies. The two deadline urgency levels are: (i) Background music, movie and game packets all have end-to-end deadlines of 200 ms, control packet has end-to-end deadline of 20 ms, foreground game packet has end-to-end deadline of 100 ms and the end-to-end deadline decrement amount is 10 ms (ii) Background music, movie and game packets all have end-to-end deadlines of 100ms, control packet, foreground game packet end-to-end deadlines and the end-to-end deadline decrement amount are the same as in (i). The utilization of the bottleneck channel in this experiment is fixed at 95%.

Fig. 4 plots the values of F_1 against two deadline urgency levels, when no strategies are used, “delay-compensation” strategy is used, “network-based” strategy is used, and when both of them are used. It can be observed from Fig. 4 that in urgency level (ii), the “network-based” strategy achieves larger reduction of F_1 than in urgency level (i). In particular, the value of F_1 is reduced by 29% in urgency level (ii), compare to 7% in urgency level (i). This can be explained as follows: in urgency level (i), there has already been a significant difference between the end-to-end deadlines of the packets of the background and foreground traffic. Thus adjusting the end-to-end deadlines of foreground traffic packets give less improvement than in urgency level (ii), where background and foreground traffic packets have the same end-to-end deadlines. Consequently, we conclude that when background and foreground traffic packets have similar end-to-end deadline urgency level, “network-based” strategy provides larger improvement.

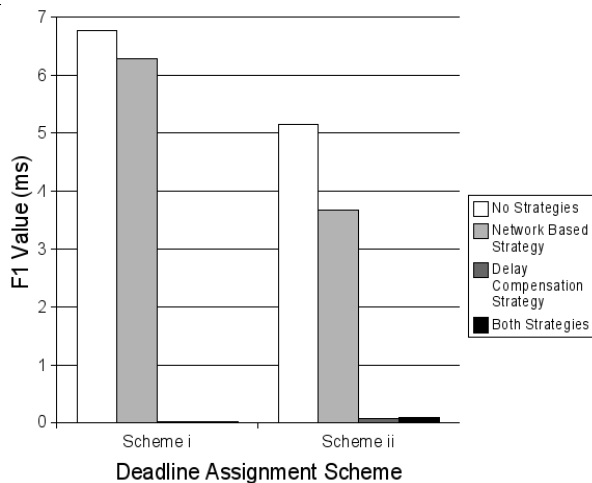


Fig. 4. Effect of Deadline Assignment Scheme on F_1

Although “delay-compensation” strategy is deadline independent by its nature, Fig. 4 shows that “delay-compensation” strategy, when used together with “network-based” strategy, is also independent of deadline urgency levels.

B. F_2 Related Results

To show the improvement on F_2 , we only show the experiment when network load level varies. Three different utilization levels on the bottleneck channel through which the foreground server-to-client game traffic goes, namely. 95%, 90% and 85% are examined. Fig. 5 plots the values of F_2 when no strategies are used and when both “delay compensation” and “network-based” strategies are used, in three different network load levels.

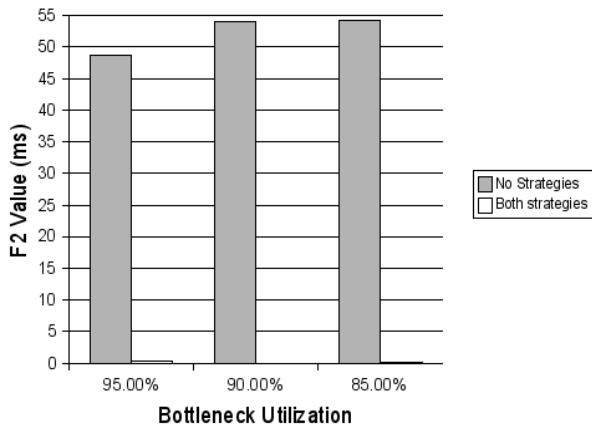


Fig. 5. Effect of Network Load on F_2

It can be observed that the combination of “delay compensation” strategy and “network-based” strategy can reduce the value of F_2 to zero regardless of the network load. In conclusion, the effect of “delay compensation” strategy and “network-based” strategy together on the value of F_2 is

network load independent; by employing these strategies, the fairness index F_2 can be significantly improved.

VII. CONCLUSION

We study the issue of fairness in games. Differ from other studies in which the fairness issue in games is discussed for the current Internet [2], [3], this research focuses on achieving fairness on deadline-based networks. The fairness in client-to-server packets and server-to-client packets are considered independently. We define fairness and develop two fairness strategies: “delay compensation” based and “network-based”. Simulation results show that using both strategies can significantly improve the game fairness in deadline-based networks.

Although it would be best to deploy the “delay compensation” strategy and the “network-based” strategy together, they can operate independently. While “network-based” strategy requires the use of deadline-based scheduling in routers, the “delay compensation” strategy can be used on the current Internet. One could deploy “delay compensation” strategy until the deadline-based network is realized.

Our experiments show that adjusting deadlines of the packets that have larger one-way delay not only helps reducing their delays but also improves the fairness of the game session. This demonstrates the capability of deadline-based network in improving fairness in FPS games. A future work could be to examine the consistency and the responsiveness of FPS games when deadlines of both foreground and background traffic are adjusted.

REFERENCES

- [1] Y. E. Liu, “Deadline based Network Resource Management”, Ph.D. Dissertation, Department of Computer Science, University of Waterloo, Waterloo, Ontario, June 2003.
- [2] C. E. Palazzi, S. Ferretti, M. Rocchetti, G. Pau and M. Gerla, “Buscar el Levante por el Poniente: In Search of Fairness Through Interactivity in Massively Multiplayer Online Games”, Proc. 2nd IEEE International Workshop on NIME 2006, 3rd IEEE CCNC 2006, Las Vegas, USA, IEEE Communications Society, January 2006, pp. 1183-1187.
- [3] K. Guo, S. Mukherjee, S. Rangarajan, S. Paul, “A fair message exchange framework for distributed multi-player games”, Proceedings of the 2nd workshop on Network and system support for games, 2003, pp. 29-41.
- [4] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, “Packet-level Traffic Measurement from the Sprint IP Backbone,” IEEE Network Magazine, Nov. 2003, vol. 17, no. 6, pp. 6-16.
- [5] P. Verissimo, L. Rodrigues, A. Casimiro, “Cesiumspray: a precise and accurate global clock service for large-scale systems”, Journal of Real-Time Systems, May 1997, No. 3, Vol. 12 (1), pp. 241-294.
- [6] H. J. Park, T. Y. Kim, S. J. Kim, “Network Traffic Analysis and Modeling for Games”. WINE 2005, pp. 1056-1065.
- [7] T. Lang, P. Branch, G. Armitage, “A Synthetic Traffic Model for Quake3”, Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology, 2004, pp. 233-138.
- [8] T. Lang, G. Armitage, P. Branch, H. Y. Choo, “A Synthetic Traffic Model for Half-Life”, Advanced Internet Architectures Swinburne University of Technology, Australia, 2003.
- [9] J. Farber, “Network Game Traffic Modeling”, Proceedings of the 1st workshop on Network and system support for games, 2002, pp. 53-57.
- [10] Abilene backbone network, <http://abilene.internet2.edu/>
- [11] S. Hacker, “MP3: The Definitive Guide”, O’Reilly, 2000.
- [12] Movie traces from Arizona State University, <http://trace.eas.asu.edu/TRACE/trace.html>
- [13] A. Le, “Fairness in Multi-Player Online Games on Deadline-Based Networks”, Honor’s Project Report, University of Manitoba, 2006.