FILA, a Holistic Approach to Massive Online Gaming: Algorithm Comparison and Performance Analysis

Stefano Ferretti\(^{(1)}\), Claudio E. Palazzi\(^{(1,2)}\), Marco Rocce\(^{(1)}\), Giovanni Pau\(^{(2)}\), Mario Gerla\(^{(2)}\)

\(^{(1)}\)Dipartimento di Scienze dell’Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40127 Bologna, Italia
Phone: +39-051-209-4503

\(^{(2)}\)Computer Science Department, University of California Los Angeles, Boelter Hall, Los Angeles CA, 90095, USA
Phone: +1-310-825-4367

\{sferrett, roccetti\}@cs.unibo.it
\{cpalazzi, gpau, gerla\}@cs.ucla.edu

ABSTRACT

The popularity of multiplayer online games has nowadays reached millions across the globe capturing the attention of both researchers and practitioners. Unfortunately, this kind of applications has still to deal with the limitations imposed by some unresolved issues. Interactivity, consistency, fairness, and scalability are the major requirements that need to be efficiently addressed in order to provide an appealing product to a huge number of potential customers all over the world. Answering to this demand, we propose holistic approach able to exploit the semantics of the game to satisfy the aforementioned requirements. In particular, we here compare different versions of this mechanism with the traditional local lag scheme. We provide extensive results also demonstrating how our scheme efficiently copes with elevated game traffic level.

Categories and Subject Descriptors
K.8.0 [Computing Milieux]: Personal Computing — Games

General Terms

Keywords
Multiplayer Online Games, Fairness, Interactivity, Consistency, Scalability.

1. INTRODUCTION

A Massively Multiplayer Online Game (MMOG) can be defined as a computer game able to support a multitude of players which interact each other within the same virtual world, across the Internet, and regardless of their geographical locations.

MMOGs trace their roots to the late ’70s, when the popularity of the Multi-User Dungeon (MUD), a text based role playing adventure, escalated from the Essex University to all-over the world [1]. Since then, MMOGs have evolved embodying now an ample class that includes several different kinds of games (e.g. car racing, first person shooter, adventure, role play game, strategic).

Indeed, in the past few years, the popularity of MMOGs has exponentially increased over the Internet [2]. Nevertheless, Internet latencies have bounded this kind of application to typically present itself as a slow-paced game, utilizing the client-server paradigm for the network communication, and efficiently engaging players only when limited in number and located in the same region where the server is positioned.

The task of providing a pleasant experience to customers becomes apodictically more challenging when trying to deploy a large scale and highly interactive online game. Indeed, the prominent networking key factors in developing MMOGs are represented by:

- Interactivity (or responsiveness) degree in the game event exchange;
- Consistency of the game state view among all the engaged players;
- Fairness ensured in terms of guaranteeing the same possibility of victory to all the players regardless of their subjective network conditions;
- Scalability in the number of contemporary players as well as in their geographical distribution.

Regarding the last point, it should be noticed that the interest of companies in online gaming emerges from the huge revenues that may be generated by a very elevated number of customers. Besides, humans are social beings which enjoy the presence of others in most of their amusement activities (i.e. team sports, movies in theatres) and the competition in challenging their skills against real adversaries.

Therefore, in order to ensure success to a MMOG, every time a new scheme is proposed as a solution for one of the first three key factors, scalability should be ensured (and verified) as well.

Generalizing this concept, developers should follow a holistic approach when designing a new MMOG, considering the whole set of requirements and aiming at the intersection of their solutions. Addressing only one among the aforementioned
requirements, in fact, could produce the unexpected and undesired result of jeopardizing the others.

Aimed at sustaining interactivity and fairness whilst preserving consistency, we have recently proposed Fairness and Interactivity-Loss Avoidance (FILA): a novel scheme able to facilitate fairness by aiming at increasing the interactivity degree [9, 26]. By doing this, FILA contradicts the general belief that interactivity and fairness would be incompatible requirements in MMOGs. We obtained this result by exploiting the semantics of the game and discarding obsolete events with a probability that depends on the current interactivity degree of the system and on some parameters.

We are now showing how our mechanism can be adapted to also satisfy the scalability requirement. In particular, we demonstrate here that FILA is particularly able to cope with intense game traffic generated by a multitude of players sharing the same virtual arena.

The remainder of the paper is organized as follows. Section 2 discusses MMOG fundamental requirements. Section 3 presents a scalable gaming architecture and compares it with other traditional architectures for online games. Section 4 provides the basics of the FILA approach. Section 5 describes the four algorithms evaluated in this work. Section 6 explains the simulative environment we have used for our experiments. Section 7 presents the experimental results. Finally, Section 8 concludes this work.

2. PROBLEM STATEMENT

The four key requirements listened in Section 1 cannot be considered as independent from each other. Aiming at improving only one of them, the others may be found negatively affected. Before evaluating any new algorithm for MMOGs, we have hence to deeply understand the tradeoff relationship that exists among interactivity, consistency, fairness, and scalability.

In particular, interactivity refers to having small delays between the generation of a game event and the time at which all the clients visualize that event. Indeed, every class of game is featured by a peculiar (and fixed) Game Interactivity Threshold (GIT) that represents the maximum delay endurable before visualizing a game event on players’ screens if one wishes to preserve interactivity. The typical GIT for fast paced games (i.e. vehicle racing, first person shooter) corresponds to 150-200ms but this value can be increased to even seconds in case of slow paced games (i.e. strategic, role play game) [6, 17-20].

If we call \( t^{\text{g(e)}} \) the generation time of event \( e \) and \( t^{\text{v(e)}} \) the visualization time of the same event at player \( i \), then interactivity is preserved at \( i \) during the delivery of \( e \) when the following condition is satisfied:

\[
t^{\text{v(e)}} - t^{\text{g(e)}} \leq \text{GIT}.
\]  

Both consistency and fairness regards having the same and contemporary game state view in all the nodes of the system. Therefore, the same class of techniques is used to achieve each of them (or both). Indeed, the easiest way to guarantee consistency and fairness is to make the game proceeding through discrete locksteps [14]. In other words, the game evolves by marching in step and players have to wait their turn before making an action. Obviously, this scheme cannot be applied to interactive games.

To ensure fairness (and consistency) in continuously evolving games several studies propose schemes based on the introduction of artificial delays in order to contemporarily visualize game events on all the players’ screens [4-8]. This set of solutions is usually referred to as local lag algorithms.

With local lag, game advancements are delayed for a sufficient amount of time in order to guarantee that all the clients in the system process and perceive the generated game events at the same time and in the same order. Indeed, since the generation time of each event is unique and considering \( C \), the set of clients, we can say that we have event-related fairness [9] for event \( e \) if condition (2) is satisfied, simply stated, if there is a unique \( t^{\text{v(e)}} \) value for all the players:

\[
t^{\text{v(e)}} = t^{\text{v(e)}} \quad \forall i \in C.
\]  

Since a single game event experiences different overall delays \((\text{OD})\) in its paths from the source to all the diverse destinations, different amounts of artificial delay \( \delta \) should be added in order to contemporarily visualize the same event \( e \) on all the players’ screens and hence to satisfy the following condition:

\[
\text{OD}_i(e) + \delta_i(e) = t^{\text{v(e)}} \quad \forall i \in C.
\]  

A possible value typically chosen for the unique \( t^{\text{v(e)}} \) is represented by the highest \( \text{OD} \) in transmitting events amongst nodes. When the highest \( \text{OD} \) is greater than GIT, however, fairness is preserved at the cost of jeopardizing interactivity for all the players. Conversely, if we use GIT as an upper bound to \( t^{\text{v(e)}} \), then we can guarantee interactivity but not fairness.

Consequently, in order to maximize the possibility to obtain both interactivity and fairness \( t^{\text{v(e)}} \) should be set as

\[
t^{\text{v(e)}} = t^{\text{g(e)}} + \text{GIT}.
\]  

Yet, the \( \text{OD}(e) \) experienced by an event \( e \) when it finally reaches client \( i \) is composed by several delay components, respectively: physical latency \( l_i(e) \), queuing time \( q_i(e) \) on routers along the path and on game servers, and processing time \( p_i(e) \). Therefore, \( \text{OD}(e) \) can be written as

\[
\text{OD}_i(e) = l_i(e) + q_i(e) + p_i(e).
\]  

Therefore, even when the network latency would allow having values of \( \text{OD} \), and hence also of \( t^{\text{v(e)}} \), lower than GIT, a large number of players, generating a huge amount of traffic, may raise the value of the other two components, thus leading us again to the crossroad between fairness and interactivity.

To conclude, the efficiency and applicability of popular, delayed-based algorithms, like local lag for example, strongly depend on the network conditions and on the interactivity degree required by the game. Yet, guaranteeing both interactivity and full fairness
through local lag can sometimes be achieved only at the cost of limiting the scalability of the game by bounding the number of contemporary participants and the geographical extension of the target player market.

It hence becomes evident as MMOGs require the use of architectural solutions and algorithms able to reduce the delay components in (5) in order to find the most efficient tradeoff among interactivity, consistency, fairness, and scalability.

3. ARCHITECTURAL SOLUTIONS

Typically, network architectures supporting MMOGs can be distinguished based on three main categories: centralized client-server, fully distributed, and mirrored game server. The centralized client-server architecture represents the simplest solution for authentication procedures, security issues, and consistency maintenance [10]. Moreover, assuming to have \( N \) contemporary players, the generated messages are in the order of \( O(N) \). On the other hand, the unique bottleneck limits the efficiency and scalability of this architecture.

Fully distributed architectures (as peer-to-peer solutions) spread the traffic load among many nodes and result in more scalable and failure-resilient systems [5]. However, identical copies of the current game state need to be stored at each node; this requires some complex coordination scheme among peers able to guarantee the coherence of all game state views. Moreover, with fully distributed architecture, IP multicast should be employed to reduce the bandwidth requirements, but multicast technology is neither generally available nor mature enough for the peculiar application we are here considering. The exchanged messages could hence raise to the order of \( O(N^2) \). Finally, authentication, cheating, and general consensus among all the peers are harder to be addressed than when a centralized architecture is employed.

Mirrored game server architecture represents a hybrid solution which efficiently embraces all the positive aspects of both centralized client-server and fully distributed architectures [12]. In this architecture, Game State Servers (GSSs) are interconnected in a peer-to-peer fashion over the Internet and contain replicas of the same game state view. Players communicate with their closest GSS through the client-server paradigm. Each GSS gathers all the game events of its engaged players, updates the game state and forwards it regularly to all its players and GSS peers.

The presence of multiple high performance GSSs helps in distributing the traffic over the system and reduces the processing burden at each node. In essence, it helps in reducing \( q(e) \) and \( p(e) \) in (5). Moreover, having each player connected to a close GSS reduces the impact of the player-dependent access technology (i.e. dial-up, cable, DSL) on the total delay experienced [13]. In this case, in fact, the communication among players results mainly deployed over links physically connecting GSSs, which can exploit the fastest available technology (i.e. optical fibers) to reduce latency. As a result, this architecture helps one in finding better solutions for the tradeoff among interactivity, consistency, fairness and scalability.

Other advantages in employing a mirrored game server architecture are the absence of a single point of failure, the networking complexity maintained at server side, and the possibility to easily implement authentication procedures. Even if synchronization is still required to ensure the global consistency of the game state held by the various servers, this requirement is made easier than in fully distributed architectures thanks to the lower number of involved nodes. Assuming to have \( N \) players and \( M \) GSSs, in fact, the generated game messages amount to the order of \( O(N+M) \), which is again \( O(N) \) unless considering the unlikely case of having more servers than players.

All these reasons depict mirrored game server architecture as the most appropriate in order to efficiently manage large-scale distributed games by embodying the advantages of both client-server and fully distributed paradigms.

4. FILA OVER A MIRRORED SERVERS ARCHITECTURE

FILA can be thought of as comprised of two complementary parts. The first one, enforced among GSSs, speeds up the delivery of “fresh” game events by dropping some events which have become obsolete since the arrival of more recent ones that supersede their content. Interested readers may refer to [15] for a deeper discussion on the notion of obsolescence and a way to include it in game events. The second part takes advantage of this reduced transmission time to magnify the efficiency of a local lag-type of algorithm to ensure fairness. FILA utilizes (4) to determine the visualization time of a game event; it thus ensures fairness without compromising interactivity.

To calculate the appropriate \( \delta \) in (3), \( OD \) should be determined for each player. For this reason, game events are marked at their creation with a generation timestamp and then sent to the destination: hence, they are orderable. Obviously, a global concept of time has to be maintained in the system. This can be achieved through a variety of solutions that enable the synchronization of GSSs’ physical clocks [21, 22], or by employing new technological synchronization devices such as GPS. Thanks to this, GSSs are able to monitor the \( ODs \) of their engaged players and make them available for the FILA algorithm.

The first part of FILA takes inspiration from Active Queueing Management techniques [25] and drops queued obsolete game events with a certain probability \( p_d \) when the average \( OD \) (avgOD) value increases putting at risk the interactivity of the system. The discarding probability \( p_d \) is directly proportional to \( avgOD \) and dependent on a constant \( Pmax \). Instead, the value for \( avgOD \), at iteration \( n \), is computed through the low-pass filter showed in (6), where \( w \) is a parameter that determines how close the average follows the sample trend:

\[
avgOD_n = avgOD_{n-1} + w \times (sample_n - avgOD_{n-1}).
\]

More in detail, with FILA, all the game events are regularly processed and forwarded while \( avgOD \) is smaller than an alert threshold named \( tmin \). When \( avgOD \) exceeds \( tmin \), the GSSs drop obsolete events with probability \( p_d \), with neither processing nor forwarding them. Finally, if \( avgOD \) exceeds the subsequent \( tmax \) (\( >tmin \) threshold), then \( p_d \) is set equal to 1 and all obsolete events waiting for being processed are discarded.

This stabilization mechanism succeeds in reducing \( OD(e) \) by impacting on \( q(e) \). In fact, the time spent in queue by a certain event is diminished by the spared processing time of preceding obsolete events which have been dropped with neither processing
nor forwarding them. Moreover, since only obsolete events are discarded, FILA fully maintains consistency in the game evolution [9, 23].

To explain FILA more in detail we use the clarifying help of Fig. 1 which provides the graphical definitions for some terms utilized in our explanation: $OD$, $ND$ (Network Delay), and $LHD$ (Last Hop Delay).

First of all, it should be noticed that FILA performs its operations in our explanation: $OD$, $ND$ (Network Delay), and $LHD$ (Last Hop Delay).

![Game event sent between players](image)

**Figure 1. Delay definitions.**

First of all, it should be noticed that FILA performs its operations on the GSSs. This choice helps us in maintaining a simpler control of the exploited game platform. Under that circumstance, however, for each event $e$, GSSs can compute $ND(e)$ but not $LHD(e)$. However an estimation of $LHD(e)$ is necessary in order to compute $OD$ and utilize it in our algorithm. For this reason, each GSS continuously monitors the latencies to each of its engaged players and maintains a variable named $\lambda_{GSS}$. The value of this variable represents the maximum among the latencies from the considered GSS to each of its connected clients (this set of clients is named $C_{GSS}$) and is calculated as follows:

$$\lambda_{GSS} = \max_{i \in C_{GSS}} \{ LHD_i \}.$$  

However, we cannot let some irremediably delay-affected client to excessively impact on the calculations performed by our scheme. Utilizing, in FILA, the excessively high $\lambda_{GSS}$ generated by some player connected very far away from the GSS, in fact, would result in very high $sample$ (and $avgOD$) values with respect to GIT. Consequently, FILA will increase the aggressiveness of its discarding function as perceived by all the players with no positive results: the “unlucky” player will still not be able to receive game events with delays below the interactivity threshold. For this reason, we need to consider a Delay Upper Bound (DUB) that is used by FILA to limit the impact of “unlucky” players on its algorithm. To this aim, (8) provides the formula for a fundamental parameter utilized by FILA to handle the impact of $LHD(e)$ on its algorithm:

$$\sigma = \min\{ \lambda_{GSS}, DUB \}.$$  

The usage of this parameter depends on the employed version of our scheme and is explained in Section 5. To determine DUB we rely on a heuristic that dynamically computes its value based on the general network condition during the game. Its formula is as follows:

$$DUB = GIT - \max\{ ND \}.$$  

where $\max\{ND\}$ represents the largest among the $ND$s experienced over all the connections between each GSS and the players engaged by the other GSSs.

To compute DUB, each GSS has hence to periodically determine the $ND$ that features in average the slowest of its connections with players engaged by the other servers. Then, this value has to be communicated back to all the other peers in order to allow a global knowledge of the worst $ND$ endured by each GSS. Finally, the highest among these maximum $ND$s can be univocally determined by each of the GSSs and used to determine the global DUB in the system as in (9).

The second part of FILA is simply in charge of equalizing the delay differences among players with a local lag-type scheme that appropriately computes the $\delta$ value shown in (3) so as to satisfy (4) whenever possible. We are going now to empirically demonstrate how the combination of phase one and two is effective in ensuring fairness and interactivity while allowing a scalable number of contemporary players.

5. IS FILA A GOOD SOLUTION?

We are going to compare four different protocols: regular local lag (LL), and three different versions of FILA (i.e. FILA-A, FILA-B, and FILA-C).

The first one, LL, embodies the traditional local lag scheme with no discarding mechanism for obsolete events. Even in this case, however, as for all the other compared protocols, the algorithm is not allowed to introduce artificial delays if this would result in jeopardizing interactivity (i.e. $t^{\nu(e)}$ cannot be set greater than $t^{\nu(e)} + GIT$).

FILA-A is the simplest among the three possible versions of this scheme. With this algorithm, in fact, no $avgOD$ is maintained and no $p_d$ is calculated. For coherence with the basics of the algorithm anticipated in Section 4, we could say that in FILA-A $t^{\nu(e)}$ and $t^{\max}$ are both set equal to GIT and that both $w$ and $P_{\max}$ are always 1. Moreover, at each iteration of (6), $sample$ is set equal to the current $ND(e)$.

When a GSS receives a game event $e$ from a player connected to one of its GSS peers, $e$ has still to travel from the considered GSS to the final players. For this reason, our scheme takes into consideration the various $LHD(e)$ by reducing the threshold used by the algorithm. Therefore, with FILA-A, each GSS performs normal delivery and local lag operations for each game event $e$ until the following condition holds:

$$ND( e ) \leq GIT - \sigma.$$  

Conversely, all obsolete events in queue are discarded with neither processing nor forwarding them.

Similarly, in FILA-B the estimation of the impact of the remaining $LHD(e)$ is considered by diminishing one of the
utilized threshold. In particular, we have $t_{\text{max}} = \text{GIT} - \sigma$ and $t_{\text{min}} < t_{\text{max}}$. Consequently, even with this algorithm, at each iteration of (6), sample is set equal to the current $ND(e)$.

Finally, FILA-C includes the estimation of the remaining $LHD(e)$ when generating sample. Therefore, we have $t_{\text{max}} = \text{GIT}$ and $t_{\text{min}} < t_{\text{max}}$. Moreover, sample is determined by the arrival of a new event $e$ as follows:

$$sample(e) = ND(e) + \sigma.$$ (11)

Even if similar, the three versions of FILA present substantial differences. In essence, FILA-A is and aggressive, yet late, approach where all the queued obsolete events are discarded when the former includes $e$. Consequently, even with this algorithm, at each iteration of (6), sample is set equal to the current $ND(e)$.

FILA-B and FILA-C, instead, try to avoid the loss of these two properties by preemptively discarding some obsolete game events when the delay trend increases over the alert threshold $t_{\text{min}}$. We may hence expect to witness smaller dropping rates with respect to the first algorithm since both FILA-B and FILA-C need to resort to dropping all queued obsolete events less frequently than FILA-A.

Indeed, this is a desirable property. In fact, even if obsolete events can be sacrificed, they are still part of the game visual progression. Dropping too many obsolete events could result in jerky rendering caused by imprecise interpolations of the missing actions. As a result, annoying artifacts in the game evolution may be generated.

Focusing on comparing FILA-B against FILA-C, we notice that the former includes $\sigma$ in the $t_{\text{max}}$ threshold, while the latter utilizes it to compute the sample value. Apparently, the impact of $\sigma$ with FILA-C should be smoothened by the low pass filter (6). However, since sample is used to compute $\text{avgOD}$, if sample is steadily augmented by $\sigma$, then even $\text{avgOD}$ results higher. Reminding now that the discarding probability $p_e$ is directly proportional to $\text{avgOD}$, it becomes evident how the use of (11) results in higher $p_e$ values. We hence expect to find FILA-C more aggressive than FILA-B.

6. EXPERIMENTAL ARCHITECTURE

We have simulated the deployment of five GSSs across U.S.A. positioned in optimal (market) locations and presenting communication latencies among each other reasonably chosen to provide an adequate support for a highly interactive MMOG, in terms of both distance among the various nodes and number of customers that can be served [3].

For the sake of a deeper comprehension, we have focused our attention on the event receiving aspect of a single GSS (GSS0), pretending that the other GSSs are sending events to it. However, since our scheme maintains unique values for GIT and DUB, plus a global concept of time which is shared among all the nodes in the system, the simulation outcomes can be projected over the whole system with no loss of generality.

Following the literature [16], network latencies among GSSs have been generated based on a lognormal distribution whose average was taken from repeated runs of the ping application. More in detail, game events coming from players connected to the so considered sending GSSs (i.e. GSS1–GSS4) and traveling towards GSS0 experience latencies with an average as reported in Fig. 2 and a standard deviation of 10ms. Online game literature also leaded us to utilize 200B as the average game event size [11].

In MMOGs, not all queued or old game events become obsolete during the game evolution. Their content may be significant and able to drastically alter the game state evolution [15]. Thus, not all queued game events can be discarded at a given GSS. The probability that an incoming event supersedes preceding ones, making them obsolete, was set to 90%. This represents a realistic scenario for a vast plethora of possible games (i.e. adventure, strategic, car race, flight simulator), where most of the events are just independent movements. In other words, critical game events that cannot become obsolete have to be considered only sporadically, such as during collisions or shots, and may represent even less than the 10% of the whole set of game events. A coherent and more detailed numerical demonstration for this claim can be found in [24].

![Figure 2. Game server deployment.](image-url)

As to the choice of critical parameters in FILA-B and FILA-C algorithms, we have set $w = 1/8$ and $P_{\text{max}} = 0.2$ for all the simulations. The $t_{\text{min}}$ and $t_{\text{max}}$ variables, instead, change with the GIT as explained in Section 5 and, in particular, $t_{\text{min}}$ is always 100ms smaller than $t_{\text{max}}$.

We have run several set of experiments varying the GIT from 150ms to 300ms in order to take into account different kinds of games [6, 17–19]. Moreover, the end-to-end (client-to-client) network latencies were chosen as uniformly distributed in the range from 75ms to a maximal value which has been varied from 90ms to 215ms to simulate several possible communication conditions.

Aimed at testing the scalability of our system, we analyzed different configurations with varying number of players. In particular, we considered scenarios where each sending GSS was gathering from its engaged players and forwarding to GSS0 an average of one new game event every, respectively, 30ms, 20ms, and 10ms. We call this parameter Average Inter-Departing Time (AIDT). Considering an average inter-generation time of 300ms
between two subsequent game actions generated by the same player, the above AIDT values represent from 50 to 150 contemporary players.

Each experiment was identically replicated to have a significant comparison among the outcomes of the three versions of FILA, plus the regular local lag algorithm. In [4], Zander et al. showed that lower latencies result in statistically higher mean kill rates and thus in unfairness. Consequently, we have chosen to evaluate as a fairness parameter the percentage of events that were delivered by our monitored server, GSS0, to all of its players before the globally assigned visualization time \( f(v) \). These game events had a visualization time as suggested by (4) thus achieving both fairness and interactivity in their delivery to all players.

7. RESULTS

7.1 Interactivity & Fairness

The charts in Fig. 3 show the percentage of game events that GSS0 was able to deliver to all of its engaged players satisfying both the interactivity and fairness requirements (besides, consistency was maintained as well). In particular, four different graphs are present reporting results coming from employing respectively: (a) 150ms, (b) 200ms, (c) 250ms, and (d) 300ms as GIT. The AIDT value, instead, was fixed at 30ms at each sending GSS for all the four scenarios. On the x-axis we have listed different values for the end-to-end latency which represents the average network latency for a game event reaching the farthest player connected to GSS0 and departing from a another player connected to the farthest of the other sending GSSs.

The outcomes of the three FILA versions are so similar that the three corresponding lines are overlapping in most of the configurations. In a few cases, however, FILA-C results the best performing algorithm. At the same time, it can be noticed that significantly higher percentages are ensured by each version of FILA with respect to LL over the various end-to-end latencies and GIT values.

Figure 3 (a, b, c, d). Interactivity and fairness improvement with AIDT equal to 30ms.

Figure 4 (a, b, c, d). Obsolete events dropped by FILA with AIDT equal to 30ms.

Obviously, having a higher GIT improves the efficacy of all the evaluated schemes since larger local lags can be utilized. However, LL experiences a premature performance decrease when the end-to-end latency increases even if it is still far from the GIT. Instead, FILA ensures a good fairness level for a larger set of end-to-end latencies. Obviously, in those configurations where the end-to-end latency is close to, or surpasses, GIT (end-to-end latency \( \geq 140ms \) in Fig. 3.a and end-to-end latency \( \geq 190ms \) in Fig. 3.b), all the schemes are unable to overwhelm network conditions thus achieving poor results. Even in this case, however, FILA behaves better than LL.

7.2 Dropping Game Events

FILA pays the shown better results with the drops of some obsolete events. Specifically, Fig. 4 reveals the percentage of game events which have been discarded by the various versions of FILA. However, in all the considered cases, less than 20% of the game events have been dropped. This represents an acceptable value since these events are exclusively obsolete ones.

FILA-A resorts to drop obsolete events only when interactivity and fairness have already been lost, moreover, at that point, it takes action by discarding all the obsolete events in queue. Therefore, this version of FILA presents the higher dropping percentage among the three in most of the configurations. Moreover, as expected, FILA-C behaves more aggressively than FILA-B and generally discards a higher number of obsolete events.

Results are particularly meaningful if we focus on those scenarios where the physical network latency is not irremediably high with respect to GIT. Considering the configurations when the maximal overall latency is lower than GIT by 35ms or more, in fact, we find that each FILA version always guarantees at least 84% of interactively and fairly delivered game events with less than 15% of dropped events.
In order to test the scalability, we have decreased AIDT to 20ms of AIDT, while Fig. 6 and Fig. 8 correspond to the case of the network. In particular, Fig. 5 and Fig. 7 refer to the case with generate scenarios with a higher level of game traffic present in the outcomes for four different GIT values. As one can expect, the higher the game traffic, the lower the interactivity and fairness degree provided by LL (see Fig. 5 and Fig. 6). On the contrary, not only is FILA able to manage higher traffic, but its performance actually improves when AIDT decreases. As a proof for our rationale, we can notice in Fig. 7 and Fig. 8 that the number of obsolete game events dropped by the three versions of FILA increases when decreasing AIDT. This is caused by higher avgOD values due to the increased traffic, but is also allowed by the presence of more game events in queue that FILA can exploit to drop obsolete ones.

7.3 About Scalability
In order to test the scalability, we have decreased AIDT to generate scenarios with a higher level of game traffic present in the network. In particular, Fig. 5 and Fig. 7 refer to the case with 20ms of AIDT, while Fig. 6 and Fig. 8 correspond to the case where AIDT is equal to 10ms. Again, in each figure we present the outcomes for four different GIT values.

As one can expect, the higher the game traffic, the lower the interactivity and fairness degree provided by LL (see Fig. 5 and Fig. 6). On the contrary, not only is FILA able to manage higher traffic, but its performance actually improves when AIDT decreases. This surprising result is simply explainable. Higher rates in game event transmissions result in generating larger queues at GSSs of packets that have not yet been processed. This amounts to a crucial problem for LL since $q(e)$ increases for all clients putting at risk the performance of the system without having any countermeasures. With FILA, instead, a larger queue of game events at a certain GSS represents also a resource. In fact, obsolete game events in queue can be discarded thus reducing the $q(e)$ that a subsequent event $e$ will experience in its traveling towards the various clients $i$. 

As a proof for our rationale, we can notice in Fig. 7 and Fig. 8 that the number of obsolete game events dropped by the three versions of FILA increases when decreasing AIDT. This is caused by higher avgOD values due to the increased traffic, but is also allowed by the presence of more game events in queue that FILA can exploit to drop obsolete ones.
Finally, analogously to the scenario with 30ms of AIDT and for the same reasons, even with AIDT equal to 20ms and 10ms FILA algorithms. The only case when this claim is contradicted is when the end-to-end latency is much lower than the considered GIT. In this case, since the high traffic volume in the game network, the $\text{avgOD}$ can steadily surpass the $\text{min}$ threshold of FILA-B and FILA-C making them executing some preemptive drops. At the same time, the large divergence between end-to-end latency and GIT makes the situation where $\text{avgOD}$ exceeds GIT and activates the dropping functionality for FILA-A a very rare event. However, even with these configurations, the percentage of discarded game events remains still very small for all the FILA versions.

8. CONCLUSION

Interactivity, consistency, fairness and scalability embody fundamental requirements that cannot be ignored when designing a new online game. Aside from the quality of the game, in fact, the ability of holistically sustain these requirements is crucial to determine the success of a MMOG. Unfortunately, as highlighted in this work, these requirements are featured with a tradeoff relationship that makes the contemporary achievement of all of them a hard task.

To this aim, we have designed an event delivery scheme, FILA, enforced among replicated game servers and exploiting the semantics of the game to ensure interactivity and fairness, whilst preserving consistency, through discarding some obsolete game events. Since only obsolete events are discarded, there is no risk that different dropping percentages at different servers could result in some unfairness [18].

We have provided extensive experimental results that demonstrate the efficacy of FILA with different types of games and with various client-to-client latency ranges. We have also contrasted different version of FILA exposing advantages and disadvantages of each of them. Finally, we have presented a scalability evaluation of the compared schemes by increasing the game traffic level in the system.

9. ACKNOWLEDGMENTS

This work is partially supported by the Italian Ministry for Research via the ICTP/E-Grid Initiative and the Interlink Initiative, the National Science Foundation through grants CNS-0435515/ANI-0221528, and the UC-CoRe Grant MICRO 05-06 private sponsor STMicroelectronics.

10. REFERENCES


