Enhanced Mirrored Servers for Network Games

Steven Daniel Webb Sieteng Soh William Lau
Curtin University of Technology
Department of Computing
Perth, Western Australia
+61 8 9266 7680
{steven.webb@postgrad,soh@cs,lauhow@cs}.curtin.edu.au

ABSTRACT
The Mirrored Server (MS) architecture uses multiple mirrored servers across multiple locations to alleviate the bandwidth bottleneck in the Client/Server (C/S) architecture. Each mirror receives and multicasts player updates to the others, simulates the game, and disseminates the new game state to players. However, keeping the game state consistent between mirrors in the presence of network delay, and maintaining game responsiveness requires each server in MS to simulate the game multiple times for each game update, and additional times in the event of costly rollbacks. In this paper we propose the Enhanced Mirrored Server (EMS) architecture. Like in the Peer-to-Peer architecture, EMS allows peers to exchange updates directly, resulting in a higher tolerance to delay at the mirrors. We propose using bucket synchronization in the mirrors so that each server in EMS simulates the game only once for each update and does not require rollbacks. The server disseminates updates to clients only in the event of inconsistency, and thus its outgoing bandwidth is lower than in MS. Our EMS uses cryptographic techniques to provide security equivalent to C/S, and prevents the timestamp cheat possible in MS. Our analytical analysis and simulations show the advantages of EMS over MS.

Categories and Subject Descriptors
C.2.4 [**Computer-communications networks**]: Distributed applications – *Client/server, distributed applications*.

General Terms
Algorithms, Performance, Reliability, Security.

Keywords
Architecture, cheating, client/server, mirrored servers, MMOG, peer-to-peer.

1. INTRODUCTION
Most networked games use a Client/Server (C/S) architecture, in which the server is the game authority whose tasks include: (i) receiving player updates, (ii) simulating game play, (iii) validating and resolving conflicts in the simulation, (iv) disseminating updates to clients, (v) storing the current game state, (vi) storing the offline player's avatar state, and (vii) authenticating players, downloading their avatar state, and billing. With only one centralized trusted server, keeping the game consistent and cheat free in C/S is straightforward. Unfortunately, C/S suffers from the following limitations: (i) the server's incoming and outgoing bandwidth are bottlenecks as the publisher must provision sufficient bandwidth for tasks (i) and (iv) at one location, which is an expensive re-occurring cost [12]; (ii) players geographically close to the server have an unfair advantage, as they will have lower game delay (response time) than those situated further away [6]; (iii) the server's processing power is a bottleneck, as it must simulate game play, and validate and resolve conflicts in the simulation, as well as calculating player's AoI in task (iv); (iv) redirecting updates through the server increases delay while consuming bandwidth and processing power; and (v) the server is a single point of failure for the system.

Several game architectures [1,2,6,8,11] have been proposed to address the C/S limitations. Cronin, *et al* [6] proposed the Mirrored Server (MS) architecture comprising multiple mirrored servers connected by fast private links to distribute the required bandwidth, reduce the range of client delays, and address the single point of failure. Keeping the game cheat proof in MS is straightforward assuming trusted mirrors. However, to maintain game state consistency MS requires expensive mirror synchronization that increases the cost of game play simulation, validation, and conflict resolution, deteriorating the server's processing bottleneck. Furthermore, MS is vulnerable to time cheating (discussed in Section 2). It also increases delay, as all updates must be redirected through the mirrors. There are several proposals focusing on improving MS [10,14]; however, they focus on fair and interactive event delivery, whereas we focus on minimising delay, increasing scalability, and cheat prevention. Note that fairness and interactivity may also be achieved through minimising delay.

Several peer-to-peer (P2P) architectures [1,2,8] have been proposed to address the C/S limitations. P2P is scalable as the bandwidth and processing requirements are entirely handled by the clients; hence, there is no central bottleneck. Furthermore, P2P systems are resource growing; as the number of clients increases so does the overall bandwidth and processing power of the system. Unfortunately, keeping the game consistent and cheat-free in P2P is significantly harder and more costly than in C/S or MS, as the latter utilizes trusted servers/mirrors to store the world state and to validate and authenticate all player updates [9].

NetGames'07, September 19-20, 2007, Melbourne, Australia

Cheating is a major concern in network games as it degrades the experience of the majority of players who are honest [12]. This is catastrophic for games using subscription models to generate revenue [7]. Although addressing cheating, consistency, conflict resolution, and persistency issues is simplified in C/S and MS, some forms of cheatings such as collusion and proxy/reflex enhancers are still possible [18]. Several P2P protocols [1,2,8] have been proposed to solve protocol level cheats. However, these protocols fail to address the information exposure and invalid command cheats which are prevalent in MMOG, while introducing new forms of cheating (*e.g.*, the inconsistency cheat) not possible in C/S or MS [17]. In addition, these solutions require costly distributed validation-algorithms that increase game delay and bandwidth, which is economically undesirable.

The Referee Anti-Cheat Scheme (RACS) [17] is a hybrid between C/S and P2P that allows players to exchange updates directly. As updates are not routed through a server, RACS has lower delay than C/S and MS. RACS uses a trusted referee combined with cryptographic techniques to prevent cheating. Webb, *et al* [17] shows that RACS offers the same level of cheat-prevention as C/S. Since the referee only sends updates in the event of inconsistencies or when peers cannot communicate directly its outgoing (out-) bandwidth is minimised. Furthermore, as updates are not routed through a server, players geographically far away are not disadvantaged. However, the referee in RACS receives, simulates, and validates all updates, and therefore its incoming (in-) bandwidth or processing power may create a bottleneck. Furthermore all of the bandwidth must be provisioned at one location. Although fewer AoI calculations are performed in RACS than in C/S, they are all executed by one referee. Finally, the referee in RACS is a single point of failure.

We propose the Enhanced MS (EMS) architecture that allows players to exchange updates directly. EMS improves MS since it: (i) reduces the mirror's out-bandwidth; (ii) reduces the mirror's processing requirements in performing game simulations and AoI calculations; (iii) reduces the average client delay; and (iv) provides security against the timestamp cheat possible in MS (see Section 2).

The layout of the paper is as follows. Section 2 provides the background on MS. Section 3 describes the proposed EMS. Section 4 provides an analytical and simulation evaluation of EMS. Finally, Section 5 concludes the paper. Note, "he" should be read as "he or she" throughout this paper.

2. MIRRORED SERVER (MS) SCHEME

2.1 MS Architecture

The MS architecture [4,5,6], shown in Figure 1, comprises multiple trusted servers (mirrors) deployed at geographically different locations connected via a private well-provisioned (low delay, high bandwidth, multicast enabled, lossless) network. Each mirror has its own Internet connection, and clients typically connect to their closest mirror for the lowest game delay. Each client sends every update to its local mirror (ingress mirror – I-mirror) which, in turn, multicasts it to all other mirrors (egress mirrors – E-mirrors). Then, all mirrors simulate the game world based on all client updates, and therefore are able to directly resolve inconsistencies. Finally every mirror periodically sends updates to its clients. When updating a client, mirrors use AoI filtering to reduce the update size. As mirrors only perform AoI filtering for connected clients, this processing requirement is shared between mirrors.

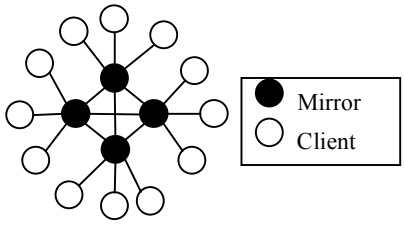


Figure 1. The Mirrored Server (MS) architecture

To reduce the bandwidth bottleneck and the range of client delays, MS distributes the responsibility of sending and receiving updates across multiple mirrors. The use of multiple servers avoids the single point of failure in C/S. Player updates in MS, like in C/S, are routed through mirrors; thus, increasing delay and consuming the mirror's bandwidth and processing power. In addition, as discussed in Section 2.2, MS requires Trailing State Synchronization (TSS) to keep the game state consistent among mirrors in the presence of network delay, and to achieve acceptable game responsiveness. TSS necessitates each mirror in MS simulate every update multiple times, and may incur multiple update disseminations and rollback steps, and thus MS has high processing overhead.

2.2 MS Synchronization

Updates exchanged between mirrors may be received at different times due to the transmission delay in the private network, resulting in inconsistencies amongst mirrors. To solve inconsistencies Cronin, *et al* [6] propose TSS to maintain the interactivity of high paced games. Unlike in C/S in which the server keeps only the current state of the game world, each mirror using TSS maintains n states: S_0, S_1, S_2, S_3, …, S_{n-1}, where S_0 is the leading state and is immediately rendered to the players screens for fast responsiveness, while the n-1 trailing states are used to resolve inconsistencies that may occur due to update delay and/or loss. Each TSS state has increasing delay behind the wall clock time. When an E-mirror receives an update from another mirror the update is simulated in all states newer than the update time. For example, consider three states S_0, S_1, and S_2 with delays of 0ms, 100ms, and 200ms respectively. A new update timestamped 125ms in the past will be immediately simulated in S_0 and S_1. If the simulation result in S_1 is inconsistent with that in S_0 a rollback occurs, else the simulation is allowed to continue. 75ms later S_2 will execute the update, and its state is compared to S_0's; if there is an inconsistency a rollback occurs, else the simulation continues. After S_{n-1}, updates are discarded and thus, any inconsistencies beyond S_{n-1} go undetected.

To rollback, the trailing state is copied to the leading state, dynamic memory structures are repaired, and all elapsed updates are re-executed; thus, any inconsistencies due to updates arriving out of order are corrected. TSS is only effective when simulating updates is inexpensive as each update is executed at least n times for TSS with n states, and more when rollbacks occur. Furthermore, performing a rollback is expensive, due to memory management costs and the need to re-execute many updates. Thus, the processing bottleneck in MS is worse than that in C/S.

2.3 MS Security

As MS utilises trusted mirrors, it is possible to achieve the same level of security as in C/S; however, the protocol in [6] is vulnerable to time cheating because updates are timestamped (for event ordering) by the untrusted clients. Consider a player P and cheater C with 25ms delay from mirror M, two states S_0=0ms and S_1=100ms, and the current game time t=1000ms. As shown in Figure 2, P sends a shoot command U_P at t=1000ms to M. M simulates U_P in S_0, calculates a hit against C, and responds with the new state S_0 to P and C at t=1025ms. C finds he has been shot and cheats by sending a dodge command U_C, with a timestamp of t=975ms at t=1050ms (the cheat). Receiving U_C, M executes it in S_0 and S_1 at t=1075. At t=1100ms M executes U_P in S_1 and detects an inconsistency with S_0 (miss vs. hit). Thus, M performs a rollback, and notifies P and C of the result S_0' (a miss) at t=1100ms. Note, this cheat is preventable by requiring each I-mirror to timestamp updates.

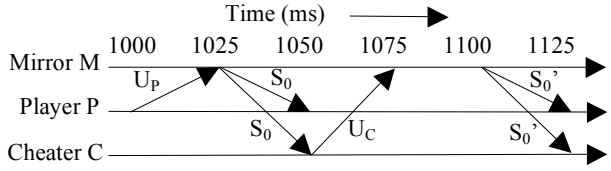


Figure 2. Time cheating in the MS architecture

3. ENHANCED MIRRORED SERVER (EMS) SCHEME

3.1 EMS Architecture and Protocols

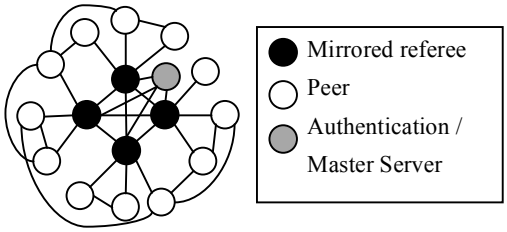


Figure 3. Enhanced Mirrored Server (EMS) architecture

The EMS architecture extends MS by allowing clients (peers) to exchange updates directly, reducing delay and the mirror's out-bandwidth and processing power. EMS also extends RACS's security measures to achieve the same degree of security as in RACS, and hence in C/S, and better than MS. EMS distributes the in-bandwidth and processing requirements across multiple referees to increase scalability and remove the single point of failure. However, EMS requires an efficient mechanism to synchronize the mirrored referees. Note that allowing P2P updates makes referee response time in EMS less critical than the mirror response time in MS, and hence, an efficient bucket synchronization [9] is sufficient for EMS, in contrast to the processing intensive TSS used in MS.

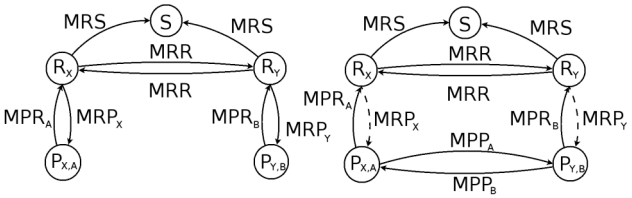
As shown in Figure 3, EMS comprises four entities: a set of mirrored referees $\{R_f \mid f$ is the unique identifier (ID) of each referee$\}$, a set of players $\{P_{f,i} \mid i$ is the unique identifier (ID) of each player connected to $R_f\}$, an authentication server S_A, and a master server S_M. An R_f is a process running on a mirror with ID f. EMS maintains game consistency and prevents cheating since each trusted/authoritative referee simulates and validates the game, and stores the current game state. A referee sends state updates only if its peers cannot communicate directly (*i.e.*, PRP mode described in Section 3.2).

Each player in EMS receives updates, simulates game play, and sends updates to his peers and his designated referee. S_A assigns a unique ID i to each player $P_{f,i}$, assigns his I-mirror f (and hence his referee R_f), authenticates joining $P_{f,i}$, downloads $P_{f,i}$'s avatar state to his host with ID i and every mirror, manages billing, and stores offline-player's avatar state. The S_A performs client to mirror assignment to minimize delay and to prevent overloaded mirrors; the algorithms used are beyond the scope of our work.

S_M divides game time into rounds of length d within which every $P_{f,i}$ generates an update and sends it to his PP players (described in Section 3.2) and his R_f, where f is the ID of its I-mirror. A late message (not received within its round) is considered for a future round assuming no newer messages have been received; otherwise it is discarded. Rounds are synchronized between hosts using NTP [8], and may be pipelined to improve responsiveness. S_M also adjusts the round length and detects inconsistent updates sent between peers (discussed later). Note that in Figure 3 we assume S_A and S_M are co-located.

3.2 EMS Communication models

As shown in Figure 4, the communication between two peers $P_{X,A}$ and $P_{Y,B}$ that are mutually aware can be through the mirrored referees R_X and R_Y (Peer-Referee-Peer: PRP mode), or direct (Peer-Peer: PP mode). In PRP each player sends and receives messages to/from his referee. In contrast, peers in PP exchange their messages directly, which reduces delay and the mirror's out-bandwidth, while maintaining security. Thus, PP is the preferable mode. In either PP or PRP mode, two mutually aware $P_{X,A}$ and $P_{Y,B}$ may or may not be using the same referee.



(a) PRP mode (b) PP mode

Figure 4. EMS communication modes

EMS considers five different message formats: (i) peer-peer message - MPP_i (U_i), (ii) peer-referee message - MPR_i (U_i, S_i, T_i), (iii) referee-peer message - MRP_f(U_i, i), (iv) referee-referee message – MRR (U_i', S_i), and (v) referee-master server message – MRS (T_i). The subscripts in MPP_i, MPR_i, MRP_f indicate each message is digitally signed by the sender (*i.e.*, $P_{f,i}$ or R_f). As in MS, we assume the private network is secure and lossless; hence, MRR and MRS messages are not signed. Note that U_i = (r, I), where r is the current round number and I is the update information; U_i' = (r', I), where r' is the current round number at the I-mirror (discussed later); S_i is secret information sent by a player $P_{f,i}$ only to his R_f; and for each update $P_{f,i}$ received from $P_{g,j}$ in the previous round T_i=$\{(j, H(U_j), D(MPP_j))\}$ where $H(U_j)$ is the hash of U_j, and $D(MPP_j)$ is the delay in receiving MPP_j. As S_i is only sent to the trusted referees, opponents do not have any secret information that may be exposed by cheating. S_i is multicast to all E-mirrors so that the simulation can be verified. T_i is forwarded to S_M to detect the inconsistency cheat and adjust the round length. By comparing the hashes in T_i, S_M detects

inconsistencies between MPPs sent between peers. Using $D(MPP_i)$, the master server calculates the optimal value for d. Note that in PP mode a mirror sends MRP only in the event of conflicts (dashed lines in Figure 4(b)).

When two PRP peers $P_{X,A}$ and $P_{Y,B}$ are within each other's AoI, their respective referees will send MRP instructing them to communicate directly; hence, transitioning to PP mode. On the other hand, $P_{X,A}$ reverts to PRP (with respect to $P_{Y,B}$) if: (i) he is no longer in $P_{Y,B}$'s AoI, and vice versa; (ii) he receives less than p percent of $P_{Y,B}$'s last $s \geq 1$ messages, or (iii) he does not receive $P_{Y,B}$'s update for more than $w \geq 0$ consecutive rounds. Reversion requirement (i) provides AoI filtering to reduce bandwidth; only players that include $P_{Y,A}$ in their AoI will be updated; requirement (ii) ensures that a minimum percentage of updates are received, preventing a cheater repeatedly sending one message and then dropping w consecutive messages; while requirement (iii) ensures that losses are not clustered, which would have a large impact on the game-play experience. For either case, $P_{X,A}$ sends an MPP_A (MPR_A) to $P_{Y,B}$ (R_X), that includes I notifying them of the reversion. R_X forwards this to R_Y, which only forwards $P_{X,A}$'s moves to $P_{Y,B}$ if $P_{X,A}$ is within $P_{Y,B}$'s AoI. Note that EMS is cheat-proof when w=0 or p=100%. The optimal values for w, p, and s should minimise PP to PRP reversions, and the number of messages that may be dropped.

When a player $P_{X,A}$ generates an update in PRP mode the following steps occur: (i) $P_{X,A}$ sends MPR_A (U_A, S_A, T_A), to his R_X (his referee running on I-mirror X); (ii) R_X receives the update, ascertains MPR_A's authenticity (see Section 4.1), and timestamps U_A with the current round number r' creating $U_A'=(r', I)$; (iii) R_X multicasts MRR (U_A', S_A) to all other referees and unicasts MRS (T_A) to S_M; (iv) all referees simulate the update and resolve state inconsistencies using r' for event ordering; and (v) all referees send the results to relevant PRP players. If a referee detects an inconsistency it will notify all relevant connected players; as all referees will detect the inconsistency all relevant players will be notified. In PP mode each referee performs all steps except step (v). The peers perform the following additional actions in steps (i), (ii), and (iv): (i) $P_{X,A}$ constructs MPP_A (U_A) and sends it to all PP peers; (ii) $P_{X,A}$ receives MPP_i from his PP peers, and ascertains their authenticity; and (iv), $P_{X,A}$ simulates and validates received MPP_i messages.

In EMS, S_M is responsible for adjusting d and detecting the inconsistency cheat. As neither responsibility prevents the game from progressing they can be located at a single central server (possibly a mirror) and do not need to be performed in real time. Requiring every referee to detect the inconsistency cheat and adjust d would increase the referee's processing requirements, and increase the traffic on the private network, without any benefit. To adjust d and detect the inconsistency cheat referees in I-mirrors unicast T_i to S_M in step (iii). As peers connected to different mirrors may be interacting, the round length must be set globally for the entire game (all mirrors and players). If S_M detects the inconsistency cheat it requests the peer to forward the offending update - via the peer's R_f - which is used to confirm the cheat using the non-repudiation quality of digital signatures.

3.3 EMS Synchronization

As in MS, EMS requires a synchronization mechanism between referees/mirrors, and we may use TSS [6] for their synchronization. However, unlike in MS, we expect most players will use PP mode in which they exchange updates directly, and thus update dissemination time in EMS is not as critical as in MS. In this paper, we use Bucket Synchronization (BS) [9] for EMS as it is more efficient than TSS.

In BS game time is divided into buckets, with all updates occurring at the same time in the same bucket. Updates in a bucket are delayed by Δ time before being executed so that all updates for that bucket are received before execution begins, synchronizing all mirrors. The advantages of EMS using BS over TSS are: (i) it needs a low processing requirement as every update is only executed once by each mirror; (ii) it needs a low memory requirement as there is only one game state; and (iii) BS does not perform rollbacks. Notice that Δ delay in BS increases the delay for PRP communications, possibly reducing interactivity. However, this performance degradation adds further incentive for peers to cooperate in PP communication. Note that TSS with a delay Δ in the leading state and no trailing states is in essence BS.

3.4 EMS Security

EMS solves all known protocol level cheats, information exposure, and invalid commands since it includes all security measures used by RACS [17]. In general, cheats are prevented by the use of signed messages (spoofing), round number r (replay attack, suppressed update, timestamp, and fixed delay), referee simulation and validation (invalid command), on demand loading (information exposure), and PRP mode (blind opponent). The recipient of an MPP, MPR, or MRP message validates its authenticity using the sender's public key. Note, r must not be used for message ordering as this would allow the timestamp cheat. EMS prevents this cheat as mirrors timestamp updates from peers, which is used by the other referees for event ordering. Message validation is performed by the referee in the I-mirror so that every MRP is only validated once, and MRR size is reduced.

In EMS, S_M detects the inconsistency cheat by comparing the hashes of all U_j in T_i forwarded by referees, and uses digital signatures to verify the cheat. In contrast, the referee in RACS is responsible for detecting this cheat as it receives all T_i [17].

Reference [1] describes the suppressed update cheat due to the use of dead-reckoning. EMS addresses this cheat as the referee's dead-reckoned state is authoritative. Thus, if a cheater drops updates he will be forced to use the referee's dead-reckoned move, which cannot be manipulated for cheating.

4. RESULTS AND ANALYSIS

4.1 Analytical evaluation

We compare the performances of MS and EMS in terms of their client's and mirror's in- and out- bandwidth requirements for N clients/peers and M mirrors. As in [4], let a and b be the number of bytes/s of player commands sent by a client to its mirror and by a mirror to other mirrors, respectively. We consider each mirror to client message comprises three components. Basic information (*e.g.*, map, time, current location) requires c bytes/s irrespective of N. On the other hand, global player information (*e.g.*, name, score) needs k bytes/s per player ($N*k$ in total), and AoI player information (*e.g.*, location, appearance) costs e byte/s per player ($e*N$ in total). We assume AoI filtering reduces the

cost of the last component by a factor of α. Table 1 shows the results, where public (private) denotes communication through the Internet (private network). Note, if all peers in EMS use PRP mode (worst case) EMS bandwidth requirements are equal to those of MS. The direct update exchanges in EMS (PP) increase the out-bandwidth requirement for each client. On the other hand, since mirrors in EMS (PP) need only send updates to resolve conflicts and provide general state information about the game, their out-bandwidth is significantly smaller than in MS. Notice that the mirror's out-bandwidth in MS grows in $O(N^2/M)$, and hence MS is not scalable when N outgrows M. In [4] MS is shown superior to C/S with respect to server/mirror bandwidth requirements, and thus we may conclude that EMS is also superior to C/S with the same respect.

Table 1. Bandwidth analysis of various architectures

		MS & EMS (PRP)	EMS (PP)
Client	**In**	$c+(k+\alpha e)N$	$c+(k+\alpha e)N$
	Out	a	$a(k+\alpha e)N$
Mirror (Public)	**In**	$a(N/M)$	$a(N/M)$
	Out	$c(N/M)$ $+(k+\alpha e)N(N/M)$	$c(N/M)$
Mirror (Private)	**In**	$bN–b(N/M)$	$bN–b(N/M)$
	Out	$b(N/M)$	$b(N/M)$

We can also analytically compare the processing requirements of TSS and BS. Note that every update in TSS with n states is processed n times, and more when rollbacks occur. Consider an update (rollback) processing time of u (v) and β the probability of an update causing a rollback. Thus, the processing cost of TSS is: $(((n * u) + (\beta * v)) * \lambda) * N$, where λ is the number of updates generated by each client per second. On the other hand, BS processing cost is only $(u * \lambda) * N$, reducing the processing of TSS by a factor of $\max(n, \beta * v)$. Since EMS uses BS, in contrast to TSS in MS, EMS greatly reduces the processing bottleneck in MS. EMS with PP mode further reduces the mirrors processing requirement as fewer AoI calculations are performed. EMS reduces the processing requirements of RACS by distributing AoI calculations between M mirrors.

4.2 Simulation

To evaluate EMS against MS we simulated both using the *Network Game Simulator* (NGS) (netgamesim.sourceforge.net) [16]. All simulations used a world size of 1000 by 1000 units, and 100 players each controlling an avatar with an AoI radius of 50 units. Avatar movement is controlled by the random-way-point mobility model with a velocity of two units per second and a wait time of 0. We simulated 1000 seconds with d=50ms (clients generate 20 updates per second). The private network delay is fixed at 50ms [6], and the peer-mirror and peer-peer delays are 200ms. Peers are evenly distributed between mirrors.

Simulation 1 compares EMS and MS in terms of their bandwidth and delay using 10 mirrors. Following [6], we considered MS using TSS with a leading state of 0ms, and trailing states: 50ms, 100ms, and 150ms with a rollback probability of 0.044, 0.006, and 0.006 respectively for each of the trailing states. On the other hand, we consider EMS using bucket synchronization with Δ=150ms, so that MS and EMS will have equal consistency and worst case delay. For EMS, we set w=6, s=200, and p=94%, as the settings are appropriate for fast paced Internet games [15,17]. These settings assume the client software can interpolate/extrapolate up to 6 consecutive lost updates, and that dropping less than 12 updates every ten seconds (94% of the previous 200 messages were received) will give a cheater an insignificant advantage [17]. If a pair of peers reverts to PRP mode, they will not reattempt PP mode for at least 60 seconds.

Figure 5 shows the average delay and the mirror/referee out-bandwidth with increasing packet loss between peers due to network loss, cheating, or firewalls. MS has nearly fixed delay and bandwidth; however, the delay and bandwidth are high as all updates are routed through the mirrors. On the other hand, with 0% loss (*i.e.*, lossless network and no cheaters) all players in EMS are in PP mode and therefore very few messages are routed through the referees; hence, the out-bandwidth and delay in EMS is far lower than MS. As packet loss increases peers revert to PRP communication, increasing the referee's out-bandwidth and average delay. Above 40% packet loss peers rapidly revert to PRP communication, dramatically increasing the impact of bucket synchronization, increasing delay. Above 70% packet loss EMS has higher delay than MS; however, the bandwidth in EMS never exceeds MS. As Internet loss rates are typically less than 1% [3], and assuming less than 69% of players are using protocol cheats or are firewalled from peers, EMS outperforms MS using this topology.

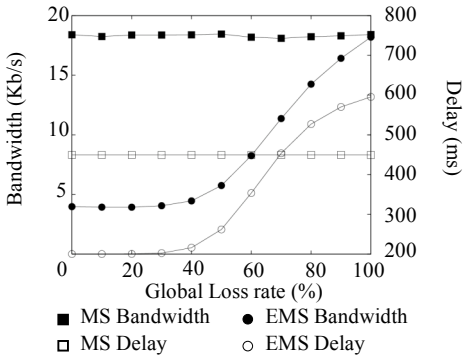


Figure 5. MS and EMS delay and out-bandwidth

To evaluate the bandwidth scalability of the referees in EMS, we repeated Simulation 1 using 20, 10, and 5 referees, each supporting 5, 10, and 20 peers respectively, in Simulation 2. As shown in Figure 6, EMS offers excellent out-bandwidth scalability when most peers use PP mode (*i.e.*, with global loss rates below 40%); the referee's out-bandwidth is minimal, as they do not forward updates. However, the referee's out-bandwidth increases as the number of PRP peers increases. Note that in either case the referee's in-bandwidth may potentially be a bottleneck.

For Simulation 3, we used EMS to compare the processing scalability and delay of BS and TSS with four trailing states. For processing time, we used the data in [6] in which every command and rollback takes 0.144ms and 1410ms, respectively. We considered processing $2*10^6$ commands (100 players, 20 updates per second, 1000 seconds) for zero to five rollbacks. As shown in Figure 7, BS requires a constant processing time (*i.e.*, $0.144ms*2*10^6=288$ seconds). On the other hand, the minimum processing time for TSS is four times that for BS (*i.e.*, 1152 seconds), and increases linearly with the number of rollbacks. It is obvious that TSS has far higher processing requirements than BS. We generated the delay results in Figure 7 by repeating Simulation 1 for EMS with BS and EMS with TSS. As shown in the figure, TSS does provide lower delay, but only when the loss

rate exceeds 40%, and the difference is marginal. However, BS offers a better user-perceived consistency than TSS affected by rollbacks.

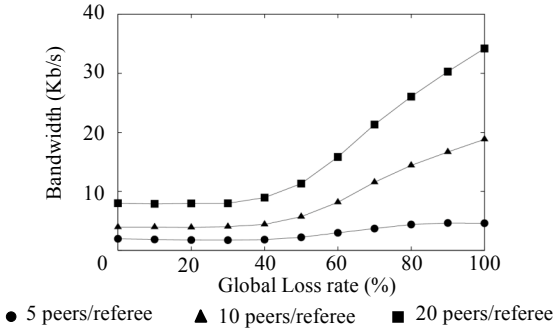


Figure 6. EMS referee bandwidth

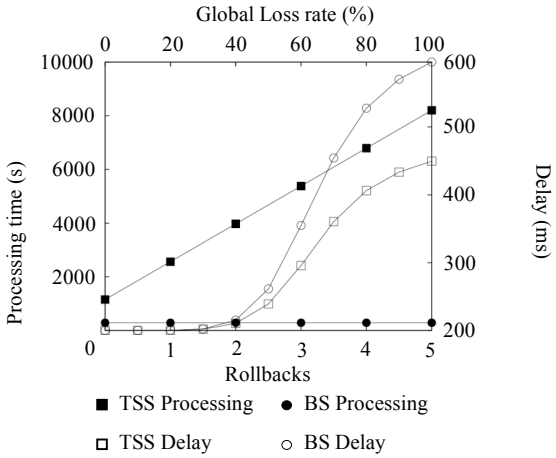


Figure 7. Synchronization processing and delay

5. CONCLUSION

In this paper we have proposed EMS to improve the performance of the MS architecture. EMS allows peers to directly exchange updates, in contrast to routing all updates through the mirrors in MS. Our simulation shows that EMS greatly reduces the mirrors out-bandwidth of MS. With no rollbacks and only one state, EMS significantly reduces the processing requirement of MS. Further, we have shown how EMS prevents timestamp cheating, possible in MS.

While EMS has high scalability in terms of bandwidth, its potential growth is still limited by its processing requirements as all mirrors must simulate the entire world. Furthermore, although the in-bandwidth is distributed across multiple referees, it may still potentially present a bottleneck. To reduce the required processing power of the referees we are investigating dividing the virtual world into regions, and load balancing the regions between referees. When the processing requirements no longer present a bottleneck we intend to distribute the referees to player machines to distribute the required in-bandwidth almost entirely to peers. However, this raises issues of referee trust and selection that must be addressed to prevent cheating.

Acknowledgements

We would like to thank the anonymous reviewers for their helpful suggestions.

6. REFERENCES

[1] Baughman, N. E., Liberatore, M., & Levine, B. N. Cheat-Proof Playout for Centralized and Peer-to-Peer Gaming. IEEE/ACM Trans. Networking 22, **1** (2007), pp. 1-17.

[2] Corman, A. B., Douglas, S., Schachte, P., & Teague, V. *A Secure Event Agreement (SEA) protocol for peer-to-peer games*. in Proc. ARES'06, pp. 34-41.

[3] Cottrell, R. L., Khan, S., *ICFA SCIC Network Monitoring Report*. http://www.slac.stanford.edu/xorg/icfa/icfa-net-paper-jan07/

[4] Cronin, E., Filstrup, B., & Kurc, A. *A Distributed multiplayer game server system*. Project Report. 2001. *warriors.eecs.umich.edu/games/papers/quakefinal.pdf*.

[5] Cronin, E., Filstrup, B., Kurc, A. & Jamin, S. *An efficient synchronization mechanism for mirrored game architectures*. In Proc. Netgames 2002, pp. 67-73.

[6] Cronin, E., Kurn, A., Filstrup, B., & Jamin, S. *An efficient synchronization mechanism for mirrored game architectures*. Multimedia Tools and Applications 23, **1** (2004), pp. 7-30.

[7] DeLap, M., *et. al.*, *Is runtime verification applicable to cheat detection?* in Proc. ACM *NetGames '04*, pp. 134-138.

[8] GautheierDickey, C., Zappala, D., Lo, V., & Marr, J. *Low-Latency and Cheat-proof Event Ordering for Distributed Games*. in Proc. NOSSDAV '04, pp. 134-139.

[9] Gautier, L., Diot, C., & Kurose, J. *End-to-end transmission control mechanisms for multiparty interactive applications on the Internet.* in Proc. *INFOCOM '99*, 3, pp. 1470-1479.

[10] Guo, K., Mukherjee, S., Rangarajan, S., Paul, S. *A Fair Message Exchange Framework for Distributed Multi-Player Games*. In Proc. Netgames 2003, pp. 29-41.

[11] Kabus, P., Terpstra, W., Cilia, M., & Buchmann, A. *Addressing cheating in distributed MMOGs*. in Proc. *NetGames '05,* pp. 1-6.

[12] Mulligan, J., & Patrovsky, B. *Developing Online Games: An Insider's Guide*. 2003: New Riders Publishing.

[13] Monch, C., Grimen, G., & Midtstraum, R. *Protecting online games against cheating*. In Proc. Netgames 2006.

[14] Palazzi, C., Ferretti, S., Cacciaguerra, S., & Roccetti, M. *On maintaining interactivity in event delivery synchronization for mirrored game architectures*. In Proc. GlobeCom 2004, pp. 157-165.

[15] Valve, *Source Multiplayer Networking*. http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking, Dec. 2006.

[16] Webb, S. D., Lau, W., & Soh, S. *NGS: An Application Layer Network Game Simulator*. in Proc. Australasian conf. IE'06, pp. 15-22.

[17] Webb, S., Soh, S., & Lau, W. *RACS: a Referee Anti-Cheat Scheme for P2P gaming*. in Proc. NOSSDAV'07, pp. 34-42.

[18] Yan, J. *Security Design in Online Games*. in Proc. *IEEE ACSAC '03*, pp. 286-295.