Round length optimisation for P2P network gaming

Steven Daniel Webb & Sieteng Soh
Department of Computing
Curtin University of Technology
Kent Street, Bentley, WA 6102
Australia
Email: {steven.webb@postgrad.,S.Soh@}curtin.edu.au

***Abstract*— The Referee Anti-Cheat Scheme (RACS) increases the scalability of Client/Server (C/S) games by allowing clients to exchange updates directly. Further, RACS maintains the security of C/S as the trusted referee (running on the server) is the game authority, simulating all client updates to validate the simulation. In RACS time is divided into rounds, and every player generates one update per round. The round length d is bounded by d_{max} which is specified by the game developer. The referee may reduce d to increase game responsiveness for players. Existing approaches to adjust d require purely distributed algorithms as they do not have a trusted central authority. These algorithms are slow and use considerable bandwidth. In this paper we propose a delay model for RACS, and two centralised algorithms to calculate d for maximum responsiveness - an optimal brute force approach and an efficient voting algorithm. We use simulation to show that the voting algorithm produces nearly optimal results, and analytical analysis to show that its processing requirements are far lower than the brute force approach.**

I. Introduction

Network games are computer games played amongst multiple players on different hosts across a network, often the Internet. Massively Multiplayer Online Games (MMOG) differ from traditional network games as they present a single universe in which thousands or tens of thousands of players participate simultaneously. Furthermore, these worlds are persistent; hence, the state of the world evolves even when the player is offline. Therefore, in addition to addressing game consistency, responsiveness, and cheat-free requirements, one must also address game persistency, system scalability, and system reliability when developing an MMOG [1].

The vast majority of 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 centralised trusted server, keeping the game consistent, persistent, and cheat free in C/S is straightforward. Unfortunately, C/S suffers from the following limitations: (i) bandwidth scalability - the server's incoming and outgoing bandwidth is a bottleneck as the publisher must provision sufficient bandwidth at one location, which is an expensive re-occurring cost [2]; (ii) processing scalability - the server's processing power is a bottleneck, as it must handle tasks (ii) and (iii), as well as calculating player's Area of Interest (AoI) [3]; (iii) responsiveness - redirecting updates through the server increases game delay; (iv) reliability - the server is a single point of failure for the system; and (v) fairness - players geographically close to the server have an unfair advantage, as they will have better responsiveness than those situated further away [4].

Several peer-to-peer (P2P) architectures [3], [5], [6] 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, as the latter utilises trusted servers to store the world state and to validate and authenticate all player updates [7].

Cheating is a major concern in network games as it degrades the experience of the majority of players who are honest [2]. This is catastrophic for games using subscription models to generate revenue [8]. Although addressing cheating, consistency, conflict resolution, and persistency issues is simplified in C/S, some forms of cheating such as collusion and proxy/reflex enhancers are still possible [9]. Several P2P protocols [3], [5], [6] 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 [10]. In addition, these solutions require costly distributed validation algorithms that increase game delay and bandwidth.

The Referee Anti-Cheat Scheme (RACS) [10] is a hybrid C/S and P2P architecture that allows players to exchange updates directly, minimising delay. RACS uses a trusted referee combined with cryptographic techniques to provide cheat prevention equivalent to that in C/S [10]. Since the referee only sends updates in the event of inconsistencies or when peers cannot communicate directly its

outgoing bandwidth is minimised. Furthermore, as updates are not routed through a server, players geographically far away are not disadvantaged.

In RACS and many other network gaming protocols [6], [5], [11] time is either implicitly or explicitly divided into rounds of length d, with every player generating one update per round. The round length is equal for all players. Reducing d increases game responsiveness; however, the bandwidth used increases. Increasing the round length reduces the bandwidth usage, but also reduces game interactivity. In many networked games there is a real-time delay constraint d_{max} beyond which the game becomes un-playable. The round length must not exceed this level (*i.e.,* $d \leq d_{max}$); however, if players have low delay the optimal d - the value of d that maximises responsiveness for the greatest number of players - may be far lower than d_{max}. In [10] we briefly discussed methods to determine the round length. In this paper we propose two algorithms to calculate the round length.

The remainder of the paper is organised as follows. In Section II we discuss related work and RACS. Section III formally describes the problem. Section IV discusses the brute force and voting algorithms to calculate the round length. Section V uses simulation to compare the optimality of the both algorithms, and Section VI concludes our paper. Note, "he" should be read as "he or she" throughout this paper.

II. Background

A. Related work

In networked computer games the game state is the current information about all avatars, monsters, items, *etc.* In this paper we assume the game state is stored on a centrally located trusted server, as this is applicable to the majority of games [12]. Every client contains a copy of a subset of the game state, corresponding to the player's Area of Interest (AoI), which is the region surrounding the player's avatar that he can perceive. Due to network delay, inconsistencies occur between the client's state and the server's authoritative state [13]. When inconsistencies occur the client may perceive unusually game behaviour, that negatively effects their game experience and performance. If clients have low delay reducing d decreases inconsistencies; hence, improving the player's experience.

For competitive networked computer games - of which there are many - it is important that the game is fair for all players. If a player has a significantly higher delay than most of his opponents he will suffer a greater number of inconsistencies; hence, he will be disadvantaged. Note that fairness is not a binary state, but a property of a game. To increase game fairness the server could artificially increase the delay of other players; however, if the slowest player has very high delay this will significantly impact on the enjoyment of all other players [13]. To be enjoyable by all players a game must be playable and fair. If network conditions are poor this may not always be achievable. At the time of writing we are not aware of any commercial games that artificially inflate player delays to improve fairness. If a player has significantly higher delay than his opponents he is responsible for attaining better Internet access.

In our work we seek to maximise the responsiveness and fairness of the majority of players. Therefore, players with delay significantly higher than the majority of players may suffer increased delay, as we aim to maximise the responsiveness for the majority of players. Further, players with delay far below the majority of their opponents may not receive optimal responsiveness, increasing the fairness and responsiveness for the majority of players.

The New Event Ordering (NEO) protocol [6] and its extension, the Secure Event Agreement (SEA) protocol [5], are P2P protocols designed to prevent protocol level cheats; however, they ignore two prevalent forms of cheating: information exposure (IE) and invalid commands (IC) which are preventable in C/S and RACS. NEO/SEA divide time into rounds of fixed length d. Players commit to a move by transmitting a secure hash of their update to the majority of opponents within d. In the following round the corresponding update is transmitted, and validated using the hash. If the committed move (the hash) is not received by the majority of players within d, or the revealed move does not match the hash, the update is discarded. NEO/SEA performs several additional checks and counter measures to prevent cheating; however, as there is no trusted game authority NEO/SEA is still vulnerable to the IE and IC cheats [10].

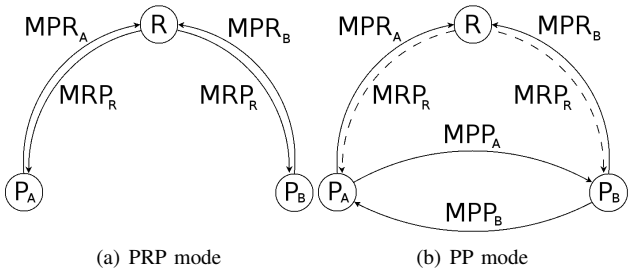
To increase responsiveness the players in NEO/SEA use a voting mechanism to increase/decrease d. Periodically players may vote to either increase, decrease or leave unchanged the round length. A *Multiplicative Increase/Additive Decrease* (MIAD) scheme is used to adjust d within a pre-defined limit d_{max} set by the game publisher. As there is no trusted authority to tally votes, all votes are transmitted to all other players requiring $O(n^2)$ messages, where n is the number of players in the group. After several rounds of voting the round length will converge.

NEO/SEA increase responsiveness by using pipelining, which allows multiple updates generated at different times to be in transit simultaneously. For example, updates may be generated once every 50*ms,* with a round length of $d = 200ms$, resulting in a pipeline depth of 4 (four updates in transit simultaneously).

B. Referee Anti-Cheat Scheme

1) Concept and protocol: RACS comprises three entities: an authentication server S_A, a set of players $P = \{P_i \mid i$ is the unique identifier (ID) of each player$\}$, and a referee R. The authentication server is used to store offline-player's avatar state, authenticate joining P_i, download P_i's avatar state to his host and R, and billing. The S_A assigns the unique ID i to each player P_i. Each player receives updates, simulates game play, and sends updates to his peers and the referee.

The referee R is a process running on a trusted host that has authority over the game state. The referee validates and resolves conflicts in the simulation to prevent cheating



(a) PRP mode (b) PP mode

Fig. 1. RACS Communication models

and stores the current game state to maintain consistency. For these tasks, it receives and simulates all updates. The referee only sends updates to peers if they are unable to communicate directly (due to high delay, message loss, firewalls, or cheating).

The referee divides time into rounds of length $d \leq d_{max}$; the developer sets d_{max} such that the game is playable. The referee R is responsible for adjusting d to maximise responsiveness for the greatest number of players. Note that decreasing (increasing) d will increase (decrease) R's outgoing traffic.

For each round r of length d, every P_i generates a pair $U_i = (r, I)$ to be included in his messages transmitted to R and other peers. Here, I is the information containing P_i's actions (*e.g.,* move, attack, *etc.*) and/or information about connections with his peers (*e.g.,* informing R about disconnecting from an opponent). To increase responsiveness rounds are pipelined; thus, updates are generated every $f \leq d$ milliseconds. The referee initialises the round number $r = 1$. Each copy of r (kept in R and each P_i) is independently incremented for every elapsed f. One can use NTP [14] for synchronising rounds.

As shown in Figure 1, RACS considers three different message formats: (i) peer to peer message – $MPP_i(U_i)$, (ii) peer to referee message – $MPR_i(U_i, S_i, T_i)$, and (iii) referee to peer message – $MRP_R(U_i, i)$, each of which is signed by the sender (*i.e.,* P_i or R). Secret information S_i is only transmitted to the referee to prevent opponents cheating by revealing information the player is not entitled too. In addition, MPR_i includes a set $T_i = \{(j, H(U_j), D(MPP_j))\}$ of elements containing the player ID j, update hash $H(U_j)$, and the delay in receiving the update $D(MPP_j)$, for every update received from opponent P_j in the previous round. The hash is used to prevent cheaters sending different updates to different players (inconsistency cheat), and the delay is used by the referee to calculate the optimal round length. Receiving $MPR_i(U_i, S_i, T_i)$, R forwards $MRP_R(U_i, i)$ to P_i's peers if the players are in PRP mode; otherwise (*i.e.,* in PP mode), R simulates the game and only sends MRP to relevant players when inconsistency is detected. Note that PP and PRP modes are discussed in Section II-B.2.

The recipient of each message validates its authenticity using the public key of the sender. 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. Thus RACS is more tolerable to slow players and network delay than [5], [6][3,6] which discards late messages. We assume the use of a public key infrastructure for authentication and non-repudiation [6].

2) Communication models: As shown in Figure 1, the communication between any P_A and P_B that are mutually aware (within each others' AoI) can be through the referee R (Peer-Referee-Peer: PRP mode), or direct (Peer-Peer: PP mode). In PRP each player sends MPR and receives MRP messages to/from R. This mode provides security equal to that in C/S. In contrast, peers in PP exchange their messages (MPP) directly, which reduces delay, and R's outgoing bandwidth while maintaining security. Thus, PP is the preferable mode. Note, in PP mode R sends an MRP only in the event of inconsistencies (dashed lines in Figure 1(b)).

A joining P_A first contacts the authentication server, which validates P_A (*e.g.,* his identity, subscription, banning, *etc.*), and downloads his avatar state to both his host and R. Then, R downloads the relevant game state to P_A's host, and notifies all affected players, *e.g.,* P_B; P_A is now in PRP mode. For these joining steps, we assume the use of existing player-authentication and startup protocols [15].

The referee converts mutually aware PRP peers (*e.g.,* P_A and P_B) into PP by sending MRP that instruct them to exchange MPP. On the other hand, P_A reverts to PRP (with respect to P_B) if: (i) he is no longer in P_B's AoI, and vice versa; (ii) he receives less than p percent of P_B's last $s \geq 1$ messages, or (iii) he does not receive P_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_A in their AoI will be updated. Requirement (ii) prevents 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_A sends an MPP (MPR) to P_B (R), that includes I notifying them of the reversion. Then, R only forwards P_A's moves to P_B if P_A is within P_B's AoI. Note that RACS is cheat-proof when $w = 0$ or $p = 100\%$. The optimal values for w, p, and s should (i) minimise PP to PRP reversions, and (ii) minimise the number of messages that may be dropped. Optimal value for w, p, and s are game dependent; thus, we do not cover them in this work.

C. Enhanced Mirrored Servers

The Enhanced Mirrored Server (EMS) architecture [16] is a combination of RACS and the Mirrored Server [4] architectures, that uses multiple mirrored referees at different geographic locations to increase bandwidth scalability. All referees simulate all updates and the entire virtual world; hence, they are mirrored. By distributing the referees across a large geographic location, Internet bandwidth costs are reduced as bandwidth is provisioned at different locations. As peers connected to different mirrors may be interacting, the round length must be set globally for the entire game (all mirrors and players); therefore, we do

not consider the EMS architecture in this paper, as the fundamental issues are identical.

III. Problem Formulation

A. Delay Model

Let $d_{i,R}$ and $d_{i,j}$ denote the average delay between a player P_i and referee R, and any two players P_i and P_j, respectively. The referee averages the arrival time of MPR_i messages to calculate $d_{i,R}$ for every P_i. The MPR_i message format includes the delay of all MPP_j messages received by a P_i from P_j; thus, the referee can calculate $d_{i,j}$ for all peers in PP communication. The delay information is stored in a $|P| \times |P| + 1$ delay matrix D, where $D_{i,j}$ is the average $d_{i,j}$, and $D_{i,|P|+1}$ is the average $d_{i,R}$. Note, $D_{i,j} = \infty$ if P_j has not received any MPP_i (either because they have not interacted, or because they cannot communicate directly).

In many multiplayer games - particularly MMOG - a player P_i is only interacting with a subset of P, denoted $PI(i)$. This subset is further divided into players using PP communication - $PP(i)$ - and players using PRP communication - $PRP(i)$. Note that $PP(i) + PRP(i) = PI(i)$, and that in general AoI filtering will ensure that $PI(i)$ is far smaller than P. The total-player-delay of P_i using round length d, $TPD(i, d)$, is the sum of the time taken for an update generated by P_i to reach all P_j, where $P_j \in PI(i)$. Formally:

$$TPD(i,d) = d \times |PP(i)| + \sum_{j \in PRP(i)} [PRD(i,d) + PRD(j,d)] \quad (1)$$

Assuming rounds begin every f *ms* (see pipelining in Section II-B.1), the peer-referee-delay of P_i using round length d, $PRD(i, d)$, is d if the delay between P_i and R is less than d, or $d_{i,R}$ plus the time until the next round if the delay between P_i and R exceeds d. Formally:

$$PRD(i,d) = \begin{cases} d & D_{i,|P|+1} \leq d \\ f \times \left\lceil \frac{D_{i,|P|+1}}{f} \right\rceil & otherwise \end{cases} \quad (2)$$

The total-system-delay for a given round length d, $TSD(d)$, is the sum of all total-player delays and player-referee delays. Formally:

$$TSD(d) = \sum_{i \in P} [TPD(i,d) + PRD(i,d)] \quad (3)$$

B. Problem Statement

The goal of our research is to minimise equation 3, thus producing the maximum responsiveness for the majority of players. The problem is thus, given matrix D, calculate the optimal value for d such that equation 3 is minimised. Note, that the referee stores the entire matrix D, whereas every P_i stores only D_i, the i^{th} row in D.

Figure 2 contains two example topologies and their corresponding delay matrices. Topology 1 assumes a tight cluster of players with a high delay to the server, such as a group of friends in Perth playing on a server located in the US. Topology 2 contains a tight group of players and the referee, and one player having far higher delay. This scenario may occur if the majority of players have DSL/Cable Internet access, but one player is connecting using a 56k modem. Intuitively, in topology 1 we believe the round length should be 64*ms*, as this will maximise responsiveness. Due to the delay between the players and the referee some inconsistencies may occur; however, we believe the increased responsiveness will make the game more enjoyable. In topology 2, it is unrealistic to delay all players to the rate of the slowest player; therefore, we recommend setting $d = 75ms$ such that players P_1, P_3, and P_4 can maximise their responsiveness. Player P_2 can still participate, but only in PRP mode. If P_2 wishes to use PP mode he must purchase faster Internet access.

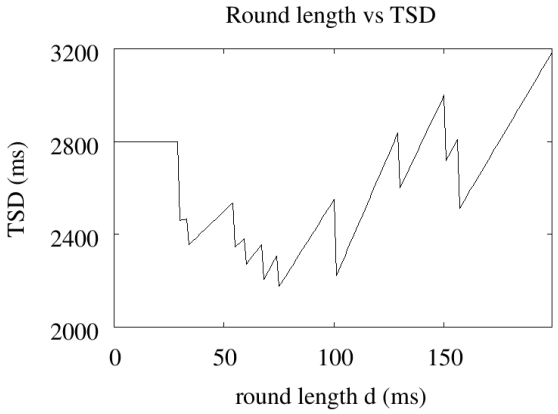


Fig. 3. TSD for topology 2.

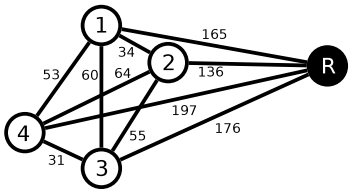
IV. Round length adjustment algorithms

A. Brute Force

The easiest approach to determine the optimal value for d is to calculate TSD for $d = [1, d_{max}]$ and select the minimum. The worst case complexity for this approach is $O(d_{max}n^2)$, where $n = |P|$. In reality due to AoI filtering it would be far lower; however, there would still be considerable processing overhead. The TSD values for topology 2 are shown in Figure 3. Note that due to the presence of many local minima, optimisation techniques such as gradient descent cannot be used.

B. Voting

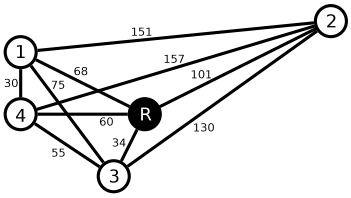
The brute-force algorithm produces optimal results; however, it has very high processing overhead and fails to maximise the benefits of the distributed nature of RACS. To reduce the processing requirements we use a voting algorithm where each player P_i votes for the d_i that minimises $TPD(i, d_i)$. The referee tallies the votes to build a Cumulative Density Function (CDF), and uses it to select the minimum d such that 50% of votes are below d. The protocol is shown in Algorithms 1 and 2. The processing requirement of the referee for this protocol is $O(n)$, as it must tally n votes.



(a) Topology 1 node placement

	P_1	P_2	P_3	P_4	R
P_1	0	34	60	53	165
P_2	34	0	55	64	136
P_3	60	55	0	31	176
P_4	53	64	31	0	197

(b) Topology 1 Matrix



(c) Topology 2 node placement

	P_1	P_2	P_3	P_4	R
P_1	0	151	75	30	68
P_2	151	0	130	157	101
P_3	75	130	0	55	34
P_4	30	157	55	0	60

(d) Topology 2 Matrix

Fig. 2. Example topologies

Algorithm 1 Round length adjustment

Instruct all peers to vote
Receive votes (d_i) from all peers
Construct CDF
Select min d such that 50% of $d_i \leq d$
Inform all peers of the new d

Algorithm 2 Vote

IMPORTS: D_j - a 1D array of delays between the the current peer P_i and all $P_j \in PI(i)$ peers and R
EXPORTS: d_i

```
min ← ∞
d_i ← 0
for every d_j ∈ D_j do
    if TPD(d_j) < min
        min ← TPD(d_j)
        d_i ← d_j
    end if
end for
```

V. Performance Evaluation

A. Algorithm comparison

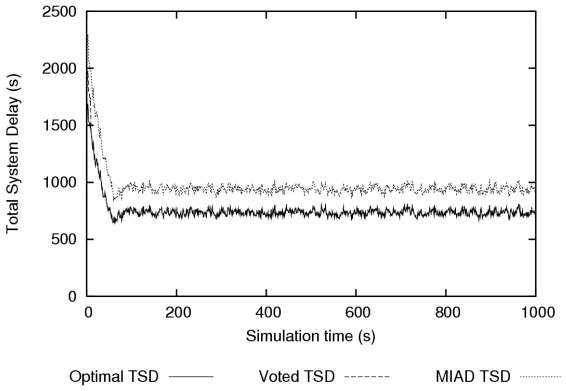
To evaluate the accuracy of the voting algorithm against the brute force algorithm we simulated both using the *Network Game Simulator* (NGS) (netgamesim.sourceforge.net) [17]. The MIAD algorithm used by NEO is included as a benchmark; doubling the round length when peers vote for an increase, and reducing d by 10% of d_{max} (20 *ms*) for a decrease. We used the *King* [18] topology to simulate delays between peers. Note, we used the average delay between peers to remove the small number of gaps in the *King* topology. A central host of the topology was selected to be the referee (a host is central if its distance from any other host is as small as possible [19]), and 1000 peers were randomly selected from the remaining 1739 hosts. We simulated a world size of 1000 by 1000 units, with each player's avatar having 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 an update frequency of 50*ms* (20 updates per second), and a maximum round length of $d_{max} = 200ms$. The round length is adjusted every 10 seconds. The Total System Delay (*TSD*) and round length (d) for each algorithm are shown in Figure 4 (a) and (b) respectively. Examining Figure 4(a), at the beginning of the simulation there is a spike in the *TSD* caused by the initial random placement of players along the edge of the world. Beyond the initial spike the difference between the brute force algorithm and the voting algorithm is so small it is not visible in the figure. As expected, the MIAD algorithm produces significantly higher *TSD*. Therefore, due to its near-optimality and low processing overhead on the referee, we recommend using the voting algorithm for round length adjustment. Furthermore, Figure 4(b) shows that the optimal (brute force) round length fluctuates far more than the voted round length. As adjusting the round length effects the responsiveness of the game the high fluctuation of the brute force algorithm will have a negative impact on the player experience. Note, players who are neighbours or team mates tend to be closer to each other in the network topology [20]; as the random-way-point mobility model does not produce this behaviour we expect the benefit of adjusting the round length will be far greater when implemented in a real game.

B. Round length adjustment frequency

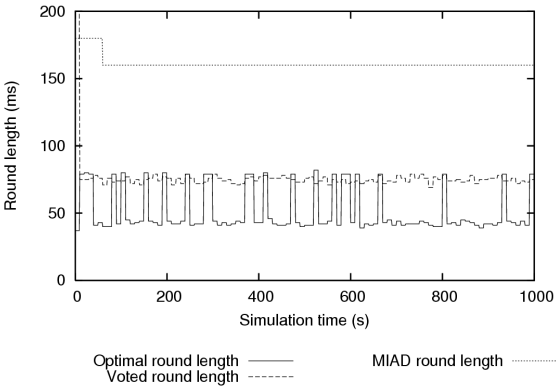
As players join and leave groups in the virtual world the round length must be updated to maintain responsiveness. Increasing the frequency of round length adjustment will increase the responsiveness to changes in group membership, reducing the *TSD*. However, excessive round length adjustment will provide little benefit, while increasing the processing and bandwidth of both the referee and peers. To demonstrate this we repeated the first simulation using the voting algorithm to adjust the round length every 1, 5, 10, 15, 20, and 60 seconds. The resulting *TSD*s for the first 100 seconds are shown in Figure 5. As expected, the more frequently the round length is adjusted the faster the TSD will converge to optimal. As there is little fluctuation in the round length beyond 100 seconds, there is little difference between the round length update frequency. As the optimal update frequency is dependent on the mobility of avatars, and the frequency of group membership changes, it is specific for every game.

VI. Conclusion

In this paper we proposed two algorithms for round length adjustment for the RACS and EMS architectures. Firstly, we developed a brute force approach that calculates the optimal round length in $O(d_{max}n^2)$, where n is the number of players. As the processing requirements for the brute force algorithm is very high when the number



(a) Total System Delay



(b) Round length

Fig. 4. Simulation 1 results

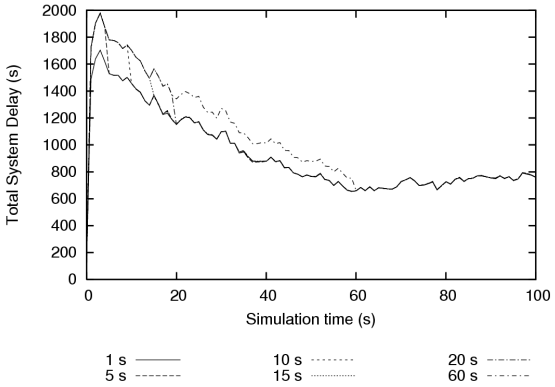


Fig. 5. Voting algorithm *TSD* at different frequencies

of players is large (such as in an MMOG), we also proposed an efficient voting algorithm with $O(n)$ complexity. We used simulation to evaluate the effectiveness of the voting algorithm against the brute force and MIAD algorithms, and found the Total System Delay (*TSD*) is only marginally higher than optimal and far below the MIAD result. Further, the voting algorithm has a lower fluctuation in round length, which would improve the game-play experience of users.

The RACS and EMS architectures require referees to simulate the entire world, and use the same round length for all players. To improve the scalability and responsiveness of RACS and EMS we intend to investigate partitioning the virtual world into regions, and load balancing the different regions between referees. Further, each region will have its own round length, increasing responsiveness.

REFERENCES

[1] S. Hu, J. Chen, and T. Chen, "VON: A scalable peer-to-peer network for virtual environments," *IEEE Network*, vol. 20, July/August 2006.

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

[3] N. E. Baughman, M. Liberatore, and B. N. Levine, "Cheat-proof playout for centralized and peer-to-peer gaming," *IEEE/ACM Trans. Networking*, pp. 1–13, 2006.

[4] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin, "An efficient synchronization mechanism for mirrored game architectures," *Multimedia Tools and Applications*, vol. 23, pp. 7–30, May 2004.

[5] A. B. Corman, S. Douglas, P. Schachte, and V. Teague, "A Secure Event Agreement (SEA) protocol for peer-to-peer games," in *Proc. ARES*, pp. 34–41, 2006.

[6] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr, "Low latency and cheat-proof event ordering for peer-to-peer games," in *Proc. NOSSDAV*, pp. 134–139, 2004.

[7] L. Guatier, C. Diot, and J. Kurose, "End-to-end transmission control mechanisms for multiparty interactive applications on the Internet," in *Proc. INFOCOM*, pp. 1470–1479, 1999.

[8] M. DeLap, B. Knutsson, H. Lu, O. Sokolsky, U. Sammapun, I. Lee, and C. Tsarouchis, "Is runtime verification applicable to cheat detection?," in *Proc. NetGames*, pp. 134–138, 2004.

[9] J. Yan, "Security design in online games," in *Proc. ACSAC*, p. 286, 2003.

[10] S. Webb, S. Soh, and W. Lau, "RACS: a referee anti-cheat scheme for P2P gaming," in *Proc. NOSSDAV*, pp. 37–42, 2007.

[11] Valve, "Source multiplayer networking." web page, Aug 2006. *http://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking* Last accessed on the 23rd October 2006.

[12] G. Huang, M. Ye, and L. Cheng, "Modeling system performance in MMORPG," *Proc. GlobeCom*, 2004.

[13] J. Brun, F. Safaei, and P. Boustead, "Fairness and playability in online multiplayer games," in *Proc. CCNC*, pp. 1199–1203, 2006.

[14] D. Mills, "Network time protocol." RFC 1305, March 1992.

[15] M. Abadi and R. Needham, "Prudent engineering practice for cryptographic protocols," *IEEE Trans. Software Engineering*, vol. 22, no. 1, pp. 6–15, 1996.

[16] S. Webb, S. Soh, and W. Lau, "Enhanced mirrored servers for network games," in *Proc. Netgames '07*, 2007.

[17] S. Webb, W. Lau, and S. Soh, "NGS: An application layer network game simulator," in *Proc. IE*, pp. 15–22, 2006.

[18] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: A tool to estimate latency between any two internet hosts, from any other internet host," in *Proc. SIGCOMM IMW*, pp. 5–18, 2002.

[19] R. Diestel, *Graph Theory*. Springer-Verlag Heidelberg, 3 ed., 2005.

[20] K. T. Chen and C. L. Lei, "Network game design: Hints and implications of player interaction," in *Proc. Netgames*, pp. 1–9, 2006.