RACS: A Referee Anti-Cheat Scheme for P2P Gaming

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

ABSTRACT
Peer-to-peer (P2P) architectures provide better scalability than Client/Server (C/S) for Massively Multiplayer Online Games (MMOG); however, they increase the possibility of cheating. Existing P2P cheat solutions only prevent protocol level cheats, ignoring two prevalent forms of cheating: *information exposure* (IE) and *invalid commands* (IC). This paper proposes the *Referee Anti Cheat Scheme* (RACS), a hybrid between P2P and C/S. As in P2P, RACS allows peers to exchange updates directly, improving its scalability. However, similar to the server in C/S, the referee in RACS has authority over the game state, providing cheat resistance equal to that in C/S. This paper describes how RACS prevents cheating – including IE and IC. Our simulation and analysis show that the average bandwidth and delay in RACS is lower than that in P2P and C/S. This paper also includes a case study of integrating RACS with a commercial network game architecture.

General Terms
Algorithms, Security.

Keywords
Cheating, client/server, MMOG, peer-to-peer, referee, security.

1. INTRODUCTION
Massively Multiplayer Online Games (MMOG) differ from traditional network games as they present a single universe in which thousands of players participate simultaneously [9]. Most MMOG use a Client/Server (C/S) architecture, in which the server is the game authority whose tasks include: T1 - receiving player updates, T2 - simulating game play, T3 - validating and resolving conflicts in the simulation, T4 - disseminating updates to clients, T5 - storing the current game state, T6 - storing the offline-player's avatar state, and T7 - authenticating players, downloading their avatar state, and billing. As the server is a trusted authority, addressing cheating, consistency, conflict resolution, and persistency issues is simplified. On the other hand, redirecting updates through the server (T1 to T4) adds game delay (response time), and consumes server bandwidth and processing power. While more servers can be provisioned, the required resources and financial costs grow rapidly with respect to the number of players, limiting C/S scalability [6,13].

Many Peer-to-Peer (P2P) architectures have been proposed to increase the scalability of MMOG [7,9], as each peer contributes its resources to perform tasks T1 to T5. However, P2P increases the possibility of cheating because it decentralises the game state to client machines.

Cheating is a major concern in MMOG [18] 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 [5]. C/S provides strong protection against cheating, as the server has authoritative control over T2 to T5. Even so, some forms of cheating cannot be detected or prevented [17]. Converting to P2P moves the responsibility of tasks T1 to T5 to the peers, making cheat detection/prevention more difficult.

Several P2P protocols [2-4,6] have been proposed to solve protocol-level cheats. However, these protocols fail to address the *information exposure* (IE) and *invalid command* (IC) cheats (prevalent in MMOG [10,14]), and introduce new forms of cheating (*e.g.*, the inconsistency cheat) not possible in C/S. In addition, the solutions require costly distributed validation-algorithms that increase game delay and bandwidth, which is economically undesirable since bandwidth is an expensive recurring cost [12]. Furthermore, the protocols in [3,6] introduce a new cheat, which we call the *undo* cheat (described in Section 2.2).

Several hybrid C/S and P2P systems [8,13] have been proposed to increase the scalability of C/S without reducing its security. Peer-to-Peer with Central Arbitrator (PP-CA) [13] has lower game delay and server outgoing-bandwidth since it allows peers to directly exchange updates; the CA receives all updates to resolve conflicts. However, PP-CA does not address cheating, and creates a new cheat (blind opponent (BO), discussed in Section 2.2). Furthermore, it does not use Area of Interest (AoI) filtering to minimise updates, a crucial component for scalable architectures.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
NOSSDAV'07 Urbana, Illinois USA
Copyright 2007 ACM 978-1-59593-746-9/06/2007…$5.00.

In this paper we propose the *Referee Anti-Cheat Scheme* (RACS) that extends PP-CA. RACS uses AoI filtering, and solves the undo, BO, IE and IC cheats. RACS requires lower delay and bandwidth than either C/S or P2P cheat solutions, while providing security equivalent to C/S.

The layout of the paper is as follows. Section 2 describes various cheats and their solutions. In Section 3 we present the details of RACS. Section 4 provides an analytical and simulation study of RACS. Section 5 concludes the paper.

2. BACKGROUND

2.1 Cheats and Solutions

We consider a cheater who can eavesdrop, replay, modify, delay, insert, and destroy messages sent/received by his host. He may also modify any program or software on his system. Note, "he" should be read as "he or she" throughout this paper. We exclude general security issues such as authentication, denial of service, *etc*.

Table 1 classifies cheats into four levels: *game, application, protocol, and infrastructure*; some cheats fall into multiple levels (*e.g.*, IE). The table extends that in [6] with the undo, BO, IE [10], IC [14], and proxy/reflex enhancers (PRE) [14] cheats. We consider PRE, such as aiming proxies, in the infrastructure level as they are deployed in the network between the client and the server [14]. Section 2.2 details the undo, IE, IC, and BO cheats; see [6] for the details of other cheats. Table 1 also shows the different cheat solutions. Note that the solutions are not mutually exclusive, *e.g.*, using C/S with PunkBuster (PB).

Table 1. Game cheats and their possible solutions

Cheat	C/S	PB/VAC2	AS	NEO/SEA	RACS
Game Level					
Bug	●		●	●	●
Application Level					
IE, IC	●				●
Bots/reflex enhancers		●			
Protocol Level					
Suppressed update, Timestamp Fixed delay, Inconsistency	●		●	●	●
Collusion					
Replay, Spoofing	●			●	●
Undo	NA		●		NA
BO	NA		NA	NA	●
Infrastructure Level					
IE	●	●			●
PRE					

Current P2P cheat solutions [2-4,6] divide time into rounds, each of which comprises two consecutive steps: (i) commit-to-an-update by transmitting an encrypted message, and (ii) reveal-the-update by sending the message key. To prevent cheating, all players must commit an update before any player reveals their key.

Lockstep [2] is too slow for many genres of games as it has a worst-case round length of *3d*, where *d* is the delay between the two slowest players. Furthermore, it is vulnerable to inconsistency, replay, and spoofing cheats [6]. Asynchronous Synchronisation (AS) [2] increases responsiveness by only requiring players with overlapping AoIs to work in lockstep. The round length in AS is reduced to between *2d* and *3d*; here *d* is the delay of the two slowest players with overlapping AoIs. Unfortunately, AS does not prevent cheaters/griefers [8] from increasing *d*. Sliding Pipeline (SP) [4] is another variation of Lockstep that increases the transmission rate by pipelining updates; however, its worst-case delay remains at *3d*.

New Event Ordering (NEO) [6] limits the round length of every group of players with overlapping AoIs to *2d*. NEO only considers an update valid if the majority of the group receives it within *d*; late updates are discarded. Each player then transmits the update's key in the second half of the round. However, NEO is vulnerable to inconsistency, timestamp, replay, and spoofing cheats [3]. Secure Event Agreement (SEA) [3] modifies NEO's cryptography to address these cheats, while maintaining its delay bound. Unfortunately, both approaches [3,6] suffer from the undo cheat (discussed in Section 2.2).

2.2 The IE, undo, IC, and BO Cheats

Information Exposure (IE): The goal of IE is to obtain secret information to which the cheater is not entitled, thus gaining an unfair advantage in selecting the optimal action. We have included IE in the application and infrastructure levels since information can be exposed at both levels, subject to how the cheat is performed. At the application level the game client or data files can be modified to reveal secret information, while at the infrastructure level, IE is achieved by modifying either (i) the graphics drivers to render the world differently [18] (*e.g.*, drawing walls transparently), or (ii) the network infrastructure to allow another host to sniff the network traffic [14]. Reference [10] proposes On-Demand Loading to address IE, at the expense of additional processing.

Undo: Let P_i denote a player with a unique identification *i*, and M_i and K_i represent a message and its key from P_i respectively. Without loss of generality, the undo cheat is illustrated in Figure 1 involving only two players: an honest P_H and a cheating P_C. Both players send their encrypted game moves (M_H and M_C) in the commit phase. Then, P_H sends key K_H in the reveal phase. P_C cheats by delaying K_C until K_H is received, and M_H is revealed. If P_C decides that his committed M_C is poor against M_H, P_C will purposely drop K_C (dashed line) *undo*ing his M_C. Even worse, in [3,6], *d* is determined by the majority of players, which

allows colluding cheaters to increase the round length d to gain time for evaluating their opponent's moves.

Invalid Command (IC): In IC, the client application or data files are modified to issue commands that originally could not be generated [14]. IC is trivially solvable in C/S since the server validates all commands. However, preventing IC in P2P is difficult because all peers (including cheaters) must agree on valid commands.

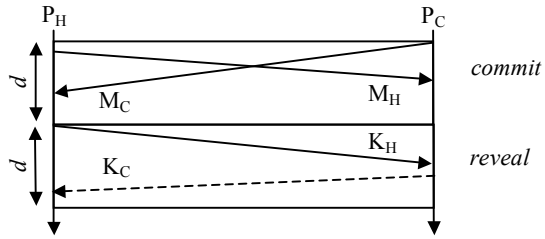

Figure 1. Sequence of messages in the undo cheat.

Blind Opponent (BO): A cheater in PP-CA may purposely drop updates to his peers (but not to the CA), effectively *blinding* them about his actions. Section 4.1 describes our RACS solution.

3. REFEREE ANTI-CHEAT SCHEME

3.1 Concept and Protocol

RACS comprises three entities: an authentication server, a set of players $\{P_i \mid i$ is the unique identifier (ID) of each player$\}$, and a referee R. The server is used to store offline-player's avatar state (T6), authenticate joining P_i, download P_i's avatar state to his host and R, and billing (T7). The server assigns a unique ID to each player. Each player receives updates (T1), simulates game play (T2), and sends updates to his peers and the referee (T4).

The referee R is a process running on a trusted host that has authority over the game state. Note that distributed referees will be addressed in future work. The referee performs Tasks T3 and T5 to prevent cheating and maintain the game's consistency. For these tasks, it receives and simulates all updates (T1 and T2). The referee performs T4 if peers are unable to communicate directly (in the event of message loss or cheating, discussed in Section 4.1).

The referee divides game time into rounds of length $d \leq d_{max}$; the developer sets d_{max} such that the game is playable. Note that R can decrease (increase) d to lower delay (reduce its outgoing bandwidth); the algorithm to compute the optimal value for d, and the frequency of using it are application dependent, and therefore we do not address these issues in this paper. In general, d should be adapted to accommodate the clients' Quality-of-Service (QoS) characteristics. We suggest setting d to the maximum client delay less than d_{max}. However, we do not recommend changing d when high priority events are occurring, as this may induce temporal disruptions.

For each round r 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). 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 d. One can use NTP [6] for synchronising d.

As shown in Figure 2, 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). It is obvious that MPP (MPR) is the smallest (largest). Note that MPP does not transmit secret information S_i (*e.g.*, health and items), which is conceptually similar to On-Demand Loading [10]. Instead, S_i is only included in MPR to the referee. In addition, he includes a set $T_i=\{(j, H(U_j), D(MPP_j))\}$ so that the referee can detect inconsistency between each hash of update U_j, $H(U_j)$, that P_i received from each of his opponents P_j (in the previous round) with that received by R. For each unmatched $H(U_j)$, R requests P_i to forward the MPP_j that he received to verify the cheat using the non-repudiation quality of digital signatures; this step prevents cheaters incriminating opponents by sending incorrect hashes to the referee. The referee uses the transmission delay of all MPP_j, $D(MPP_j)$, to adjust d. 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 3.2.

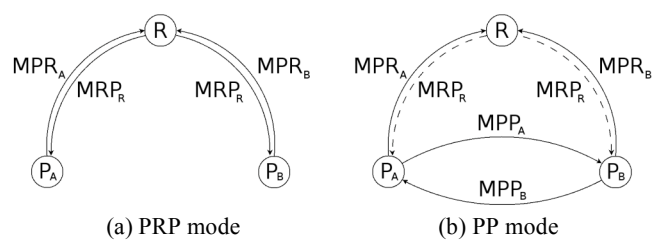

Figure 2. RACS communication models

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 [3,6] which discards late messages. We assume the use of a public key infrastructure for authentication and non-repudiation [6].

3.2 Communication Models

As shown in Figure 2, 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, R sends an MRP only in the event of conflicts (dashed lines in Figure 2(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 [1].

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.

Each leaving P_A (in PRP or PP) sends MPR (with I=QUIT), which makes R upload P_A's avatar state to the server, which in turn, sends an acknowledgement (ACK) to both P_A and R. Receiving the ACK, P_A disconnects from R, and R notifies all affected players.

Every P_i in PP mode sends his MPP (MPR) to all affected peers (R) for each elapsed d. Thus, for every round, R (each player) expects a message from each player (all other players). However, due to communication failures or cheating, a message may not arrive. Assume P_B and R are expecting a message from P_A. We consider three cases for missing messages: (i) neither receives, (ii) only R receives, and (iii) only P_B receives a message. In RACS, P_i and R dead-reckon the avatar of each P_j whose message is not received. R's state is authoritative, and it notifies affected players about inconsistencies caused by dead-reckoning. In case (i), only P_A may be disadvantaged, as P_B and R have matching state. However, in case (ii), P_B might be slightly disadvantaged if his game state is incorrect. Finally, case (iii) disadvantages both P_A and P_B since R's dead-reckoning may make their states incorrect. Note that for cases (i) and (ii), if the missing message violates reversion requirements (ii) or (iii), P_B will revert to PRP.

Every P_i in PRP mode sends MPR_i (U_i,S_i,T_i) to R for each elapsed d. In the following round, the referee sends $MRP_R(U_i,i)$ to P_i's peers. The referee (peer) dead-reckons P_i's avatar for each missing MPR (MRP); any inconsistency is resolved using the R's authoritative state.

4. RACS PERFORMANCE EVALUATION

4.1 Security

PRP mode provides security equivalent to that in C/S, since both models use a trusted entity to simulate the game and forward updates. Obviously cheat solutions in PRP are similar to those in C/S and, thus, are not discussed in this paper. In the following, we explain how RACS in PP mode addresses various cheats. Throughout our discussion, we assume a referee R, a cheating P_C, and P_C's opponent P_H.

Bugs: RACS assumes that bugs will be fixed by software patches (released by the publisher), as does C/S.

IE: P_C does not receive any secret information S_H, which is only included in MPR; thus IE is prevented.

Bots/reflex enhancers: As in C/S [5], one may combine RACS with PB or VAC2 to detect bots/reflex enhancers.

IC: R validates all player commands to prevent IC. Further, the authentication server stores all offline players' avatar state to prevent command tampering.

Suppressed update: Missing messages are dead-reckoned/ interpolated by P_H and R. Since R's state is authoritative, P_C gains no advantage from suppressing his updates.

Fixed delay: Applying fixed delay may make P_C violate reversion requirements (ii) or (iii), which, in turn, will revert P_H from PP to PRP (with respect to P_C). This will punish P_C because his delay (with respect to P_H) will be two hops, while that of P_H (with respect to the other PP peers) will be one hop. If P_C's message arrives within the round (not late), RACS does not consider it cheating. Since late updates are indistinguishable from cheating, the solution may penalize honest but slow players; nevertheless, it is better than [3,6] which prohibit slow peers from playing.

Inconsistency: R detects inconsistency by comparing the hashes of all the messages received by P_H in the previous round with those received by R from P_C (see Section 3.1).

Timestamp: Late updates are considered for the following round (Section 3.1), and thus the cheat is prevented.

Collusion: RACS does not address the collusion cheat, nor do existing P2P and C/S solutions [17].

Replay attack: A message is invalid if another message with larger r has been received (see Section 3.1); hence, r is a nonce to detect replay of old messages.

Spoofing cheat: RACS solves this cheat by authenticating the signature of the message.

Undo: RACS does not use the commit/reveal steps as in [3,6], and thus the undo cheat is impossible.

BO: This cheat is equivalent to missing message case (ii) (Section 3.2). If the cheat results in P_C violating reversion requirement (ii) or (iii), P_H will revert to PRP, which solves the cheat. Otherwise P_C gains an insignificant advantage.

4.2 Analytical Evaluation

Our analysis assumes one update (size L_u bytes) for every d=T_u seconds. We consider a game with N players, and every player is aware of at most M opponents. The worst (best) case for RACS is when no (all) peers use PP. A peer sends L_u/T_u ((L_u/T_u)*M) bytes in the worst (best) case. For both cases, the inbound traffic to the referee and each peer is (L_u/T_u)*N and $(L_u$*M$)/T_u$ bytes, respectively. Thus, the inbound bandwidth requirement of RACS is equivalent to that of C/S, and may potentially be its bottleneck. The best case outbound traffic from the referee is negligible since MRP is only sent to resolve inconsistencies. In the worst case, the referee sends $((L_u$*M$)/T_u)$*N bytes, equivalent to the server in C/S. The AoI filtering, used in RACS, greatly reduces the outgoing bandwidth of the referee and peers compared to the server and peers in PP-CA, while maintaining the same delay. Thus, RACS is more scalable than PP-CA.

RACS is superior to NEO/SEA [3,6] in several ways. First, RACS requires lower bandwidth. Every message in SEA is comparable in size with each MPR_i, and is sent to all peers in the same region. In contrast, peers in RACS send MPR_i only to the referee, and use MPP_i (smaller than MPR_i) between peers. Thus, the peer bandwidth in RACS is lower than in SEA. Second, RACS determines dynamic d more accurately and faster with lower bandwidth. Here, NEO/SEA use a distributed algorithm [6] to calculate d, in contrast to our centralized method that requires lower bandwidth cost. Further, directly setting d to the maximum client delay computes it more accurately and faster. Finally, in contrast to RACS, NEO/SEA requires complex group selection algorithms to prevent cheaters from colluding against a minority of honest players.

4.3 Simulations

We used the *Network Game Simulator* (NGS) [16] (*netgamesim.sourceforge.net*) to show the impact of cheaters on RACS delay and bandwidth requirements. We considered a game with a world of size 5000 by 5000 units, and a referee that handles 5000 players 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 for a loss-less network with no late messages. Each player generated a message every d=50ms. Further, after reverting to PRP, a pair of peers will not attempt PP mode for at least 60 seconds. We set w=0 (thus, p and s are irrelevant) to show the worst case bandwidth and delay costs of RACS in preventing cheats; equivalently, p=100% might be used. Here, the cheaters do not send MPP messages, as in BO and suppressed update. These cheats have the greatest impact on the average delay and R's outgoing bandwidth because they require the referee to forward updates (U_i in MRP) to other peers. We varied the percentage c of cheaters, randomly selected, from 0% to 100%.

Figure 3 shows the referee's average and maximum outgoing bandwidth per second (left Y axis) and the average delay (right Y axis) with an increasing percentage of cheaters. The average bandwidth (delay) was calculated by dividing the total bandwidth used (delay of all messages) by the length of the simulation (number of messages), whereas the maximum bandwidth was the peak bandwidth consumption per second measured in the simulation. As expected, the figure shows that the best case (worst case) occurs when c=0% (c=100%) as all updates are exchanged directly (routed through the referee) using PP (PRP) mode. Figure 3 shows that RACS scales well, even in the presence of cheaters, as honest players continue to exchange updates. Note that the average bandwidth and delay of RACS never exceeds those of C/S (c=100%).

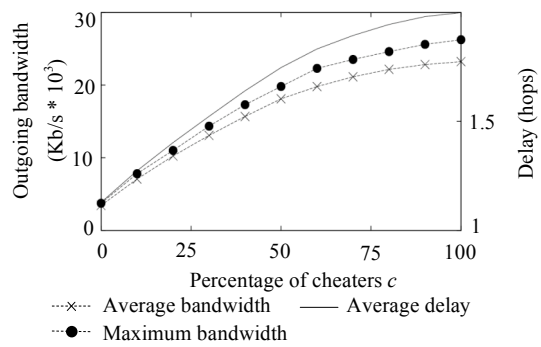

Figure 3. RACS with increasing cheaters.

The following simulation illustrates how a developer sets optimal w, s, and p for a lossy network. We consider the Source Engine (SE) in Halflife (HL) [15] to show how the values are determined. Note that HL requires very low delay [4], and thus this illustration is applicable to all genres of games, including MMOG. In the SE an update is generated every 50 milliseconds. As most client's delay exceeds this, messages must be pipelined. Note that RACS adopts a similar pipelining approach to that in [6] to avoid any security issues. The SE does not render received updates for 100ms, and uses interpolation to smooth player transitions. In the event of two consecutive message losses, the SE client dead-reckons for up to 250ms; therefore, the client halts after seven consecutive losses.

From the described specifications, we set d=50ms and w=6 ((100ms+250ms)/50ms - 1=6); hence, a peer reverts to PRP after seven consecutive losses. We believe that losing 2*w=12 messages per 10 seconds will give a cheater an insignificant advantage. Thus, s=200 (10 seconds/50ms), and p=94% ((1-12/200) * 100%). The simulation uses MPP message loss rates from 0% to 50%. We assume the communication to/from the referee is well provisioned, and therefore the loss rate for MPR and MRP is insignificant. Also, all messages arrive on time or not at all. Since the

effects of modifying w and p can only be observed when players have repeated interactions, we simulated 12 players, each with an AoI radius of 200, in a world of size 100 by 100 so that all players are constantly mutually aware. We simulated the SE with 0 (c=0%), 3 (c=25%), and 6 (c=50%) cheaters. We use all remaining parameters from the previous simulation.

Figure 4 shows the average bandwidth and delay of the simulation. The figure also includes a worst-case base line, *i.e.*, w=0, p=100%, c=0%. With no cheaters, it is obvious that increasing w and reducing p greatly reduce the referee's outgoing bandwidth and average game delay (highest *vs*. lowest plots in the figure).

The figure shows that RACS is highly tolerant to loss. Irrespective of the number of cheaters, loss rates below 20% do not impact the outgoing bandwidth and the average delay; beyond 20%, RACS performance degrades rapidly. We believe that the critical point is caused by the values of p and w, and therefore further investigation is required for tuning the parameters. However, the results show that increasing numbers of cheaters has a greater impact on performance than loss rate; as more peers revert to PRP mode. Nevertheless, the upper bound of RACS delay is two hops (as in C/S), below SEA's three-hop bound. In addition, as discussed in Section 4.2, its overall bandwidth never exceeds that in C/S and SEA.

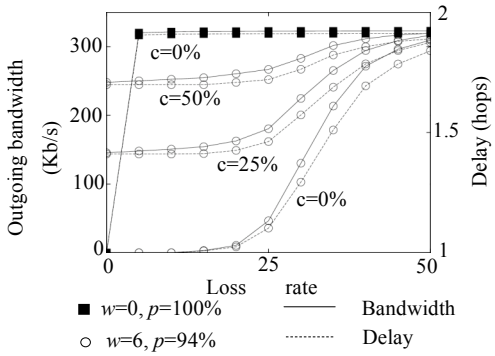

Figure 4.RACS with increasing message loss and cheaters.

5. CONCLUSION

We have extended the cheat classification in [6] and proposed RACS, which has the following benefits: (i) it provides security equal to C/S, while reduces delay and the server/referee's outgoing bandwidth; (ii) it is more effective and efficient than existing cheat solutions [2-4,6], as it is secure against the IC, IE, undo and BO cheats with lower cost; (iii) it allows peers with poor connections to play using PRP, unlike [3,6], and (iv) its centralised algorithm calculates d more accurately, faster and with lower bandwidth than the distributed algorithms in [3,6].

As with PP-CA, RACS reduces only the outgoing bandwidth; it does not address the scalability issues of incoming bandwidth and referee processing requirements. We are investigating the use of multiple referees in a server cluster (similar to Federated C/S) and/or in peers to reduce the referee's incoming bandwidth and processing requirements; hence, improving RACS scalability. Distributing referees to peers increases scalability; however, this raises issues of referee trust, selection, load balancing, and synchronization. These issues require further investigation.

6. REFERENCES

[1] Abadi, M., and Needham, R. *Prudent Engineering Practice for Cryptographic Protocols*. IEEE Trans. Software Engineering 22, **1** (1996), pp. 6-15.

[2] 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.

[3] 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.

[4] Cronin, E., Filstrup, B., & Jamin, S. *Cheat-Proofing Dead Reckoned Multiplayer Games*. in Proc. *Int. Conf Appl. Development of Computer Game*. 2003.

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

[6] 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.

[7] Hu, S. Y., Chen, J. F., & Chen, T. H. *Von: A Scalable Peer-to-Peer Network for Virtual Environments.* IEEE Network 20, **4** (2006), pp. 22-31.

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

[9] Knutsson, B., Lu, H., Xu, W., & Hopkins, B. *Peer-to-Peer Support for Massively Multiplayer Games*. in *INFOCOM '04, Hong Kong*, **1**: pp. 7-11.

[10] Li, K., Ding, S., McCreary, D., & Webb, S. *Analysis of state exposure control to prevent cheating in online games*. in Proc. ACM *NOSSDAV '04,* pp. 140-145.

[11] Li, J., & Kang, X. *mSSL: Extending SSL to Support Data Sharing Among Collaborative Clients*. in Proc. *ACSAC'05,* pp. 397-408.

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

[13] Pellegrino, J. D. & Dovrolis, C. *Bandwidth requirement and state consistency in three multiplayer game architectures*. in Proc. *NetGames '03,* pp. 52-59.

[14] Pritchard, M., *How to Hurt the Hackers*, in Game Developer Magazine, Jun. 2000. pp. 28-30.

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

[18] Yan, J. & Randell, B. *A systematic classification of cheating in online games*. in Proc. ACM *NetGames '05,* pp. 1-9.