Detecting Cheaters for Multiplayer Games: Theory, Design and Implementation[1]

S.F. Yeung John C.S. Lui
Department of Computer Science & Engineering
The Chinese University of Hong Kong
{sfyeung,cslui}@cse.cuhk.edu.hk

Jiangchuan Liu
School of Computing Science
Simon Fraser University
Vancouver, Canada
jcliu@cs.sfu.ca

Jeff Yan
School of Computing Science
University of Newcastle
Newcastle, UK,
Jeff.Yan@ ncl.ac.uk

Abstract— **Massively multiplayer game holds a huge market in the digital entertainment industry. Companies invest heavily in the game and graphics development since a successful online game can attract million of users, and this translates to a huge investment payoff. However, multiplayer online game is also subjected to various forms of hacks and cheats. Hackers can alter the graphic rendering to reveal information otherwise be hidden in a normal game, or cheaters can use software robot to play the game automatically and gain an unfair advantage. Currently, some popular online games release software patches or incorporate anti-cheating software to detect known cheats. This not only creates deployment difficulty but new cheats will still be able to breach the normal game logic until software patches are available. Moreover, the anti-cheating software themselves are also vulnerable to hacks. In this paper, we propose a scalable and efficient method to detect whether a player is cheating or not. The methodology is based on the dynamic Bayesian network approach. The detection framework relies solely on the game states and runs in the game server only. Therefore it is invulnerable to hacks and it is a much more deployable solution. To demonstrate the effectiveness of the propose method, we implement a prototype multiplayer game system and to detect whether a player is using the "aiming robot" for cheating or not. Experiments show that not only we can effectively detect cheaters, but the false positive rate is extremely low. We believe the proposed methodology and the prototype system provide a first step toward a systematic study of cheating detection and security research in the area of online multiplayer games.**

I. Introduction

In 2004, multiplayer online games generate billions revenue. The Internet Research Report - Online Games 2004 [7] shows that there are nearly 300 online game producers around the world, with 175 online games and nearly 20 million players in the year 2004. Up to February 2005, just the game "*World of Warcraft*" on its own has reached over US $200 million revenue from monthly subscription fees with more than 200,000 users playing online simultaneously.

As in all aspects in life, some players may use various forms of cheat so as to gain an unfair advantage over honest players. With the use of cheats, a cheating player may have an overwhelming superiority in terms of destroying other avatars (which are graphical representation of other players in an online game) in the virtual world. As a matter of fact, cheating in multiplayer games is becoming so common because cheats are easily accessible on the Internet. For example, Blizzard Entertainment once banned tens of thousand Diablo accounts whose players are believed to be cheaters [3].

Cheating in a first-person shooter game is especially annoying. The fun in playing a first-person shooter game is on the extensive and continuous interactions with other players, playing against a cheater is definitely an unpleasant experience since an honest player will have very little chance, if any, in beating the cheater and the cheater will eventually gain enough points and drive away most honest players from the game. Rampant cheating will destroy the entertainment value of a game, then honest players will eventually abandon the game, which implies that the game developer who invested heavily on the game development will suffer a significant financial loss.

Although many first-person shooter games are now incorporated with anti-cheating software, which are essentially pattern scanners, cheating cannot be completely prevented. Because the computer is in the hands of the cheaters, they can always work around with the anti-cheating software. Besides using the software solution, online game companies also need to employ enough people to monitor the game constantly so as to discover potential cheaters. For example, some online game servers have administrators and if they discover some suspicious players, or receive sufficient number of complaints from other players of accusing another player for foul play, then the administrator has the right to kick out a suspicious player from participating in the online game. Obviously, these types of solutions are labor intensive and they are not a scalable solution for detecting cheaters in online games that support thousands of players.

Although cheats can be implemented in many different ways and can perform differently to achieve the same purpose, these forms of cheat essentially produce similar playing patterns. In this paper, we propose an efficient and scalable solution to automatically detect whether a player is a potential cheater or not. In particular, we use the Dynamic Bayesian Network

[1]"The work of John C.S. Lui was supported in part by the SHIAE grant and the Microsoft-CUHK Joint Lab Research Grant".

(DBN) approach for our cheat detection framework. To test the effectiveness of the proposed methodology, we also develop a prototype multiplayer game server, and the cheat detection engine is enabled within the game server only. Because our framework relies solely on the ordinary game states and runs in the server side, therefore it is not vulnerable to hacks and it is a better deployable solution than the conventional software patches or human monitoring schemes. We also test our cheat detection method on a "*first-person shooter*" game for the detection of a specific cheat called aiming robot. We also test on several enhancements of the aiming robot and the results show that our framework can detect cheats effectively.

The remaining of this paper is organized as follows. In Section 2, related work on cheat detection or cheat prevention for multiplayer game is presented. In Section 3, we provide the necessary background for Dynamic Bayesian Network. In Section 4, we present the architecture of our prototype, as well as our cheat detection mechanism. We also quantify the computational complexity of our cheat detection algorithm. Experiments for showing the effectiveness and scalability of the proposed method is presented in Section 5. Section 6 concludes.

II. Related Works

There are works about cheat-controlled protocol that prevent look-ahead cheats. [1] proposed a lock-step protocol on a distributed game model wherein players will announce their decisions in cryptographically secure one-way hash as a commitment. Only when all players have announced their commitments, players then reveal their decision in plaintext. Thus a cheater cannot gain any advantage by being the last player to make decision. The authors also proposed some optimizations to overcome the synchronization problems caused. In [2], authors proposed a scheme that enforces the fair-ordering of the message delivery so that cheat will be restricted to a certain level or may even be detected. In [4], author extended the idea so that the cheat-proof protocol can be used in game with dead reckoning. In [5], authors proposed the use of runtime verification to verify game codes. This approach mainly targets on cheats that exploit implementation bugs such as trade-hack in MMORPG, but is not applicable to cheats that involve modification of client code that are loaded into memory at runtime.

III. Design and Implementation

In this paper, we present our framework of cheating detection with a real multiplayer online game. To demonstrate the effectiveness of the proposed method, we built a prototype multiplayer game system. In particular, we use the open source first-person shooter game Cube [8] as our testbed. We choose a specific type of cheat called aiming robot (or aimbot for short). This type of cheat is frequently used by various players to gain unfair advantages.

A. Dynamic Bayesian Network (DBN)

The overview of the DBN we used for our experiments is illustrated in Figure 1. In this section, we explain the details of our dynamic Bayesian network and discuss how the network can be used to detect the use of an aimbot.

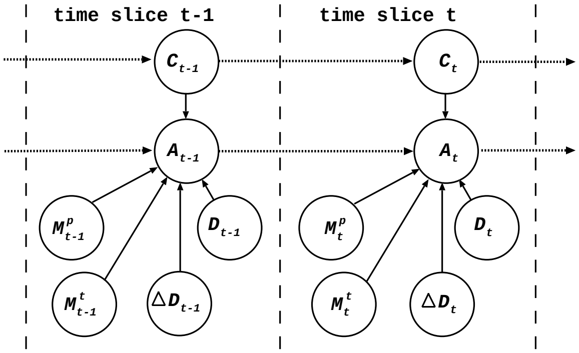


Fig. 1. The DBN used for aimbot behavior detection.

In a first-person shooter game, several state information will certainly influence the aiming accuracy of a player directly. For example, when the player is closer to the target, the higher the chance the player can hit the target. Also, it is easier to aim at a static target than a high speed moving target. Likewise, a player should have a higher aiming accuracy when that player is stationary, as compare to a player who is aiming while moving at the same time. Certainly, the use of an aimbot will affect the aiming accuracy of a player. Therefore, the probability distribution of a player's aiming accuracy, $P(A)$, is dependent on the following random variables:

1) whether the player is cheating or not, which is denoted by the random variable C,
2) whether the player is moving or not, which is denoted by the random variable M^p,
3) whether the player's aiming target is moving or not, which is denoted by the random variable M^t,
4) whether the player is changing the aiming direction, which is denoted by the random variable $\triangle D$, and
5) the distance between the player and the aiming target, which is denoted by the random variable D.

Note that these random variables themselves are not dependent on any other random variables. This is because the habit and the skill of the player and also the environment of the virtual world, such as the location of some special items, will dominate the activities of a player. It means that the input of a player is normally not dependent on other outcomes in a game session. For this reason, we model the above parameters themselves as independent random variables.

Moreover, we model the aiming process as a first order Markov process. Because aiming is a fine tunning process, once a player aimed accurately, probably only small adjustments are required to keep the accuracy. It means that the probability distribution of a player's accuracy on a certain time slice t is dependent on the player's accuracy on the previous time slice $t-1$.

Also, the change in probability of whether the player is cheating or not is also modelled as a first order Markov process. We make this dependency because a cheater may want to enable the aimbot in a time slice but disable the aimbot in the following time slice.

B. Training and Inference

All of the random variables we need to infer the probability of cheating, i.e., C, M^p, M^t, $\triangle D$ and D, can be obtained directly, or derived from the game states contained in the update packets by various players. Usually, there will be more than one visible target a player can aim. We define a player's current target as the latest target the player aimed accurately, to illustrate, let player A aimed at player B accurately, then for player A, $\tilde{Target}_A = B$ until player A aimed at another player accurately or player B becomes invisible to player A. The variables Distance (D) and Accruacy (A) are both derived against this current target. Also, for those random variables having continuous values, i.e. Distance (D) and Accuracy (A), we discretize them into finite elements. Therefore, in training the dynamic Bayesian network, we need to consider the following random variables which take on the following values:

1) Cheating (C), with $C \in [true, false]$,
2) Player Moving (M^p) with $M^p \in [true, false]$,
3) Target moving (M^t) with $M^t \in [true, false]$,
4) Changing aiming direction ($\triangle D$), with $\triangle D \in [true, false]$,
5) Distance from aiming target (D), with $D \in [0, 1, 2, 3]$ in which larger the value implies further away is the distance and,
6) Aiming accuracy (A) with $A \in [0, 1, 2, 3]$ in which the lower the value implies higher is the aiming accuracy.

Using these data, one can obtain the following prior probability distributions by counting frequencies and then normalize the values:

1) $P(C_t|C_{t-1})$, and
2) $P(A_t|A_{t-1}, C_t, M_t^p, M_t^t, \triangle D_t, D_t)$,

Inferring the probability of cheating for any particular player follows the following steps. At the very first time slice where $t = 0$, we initialize $P(\tilde{C_0} = true)$ to 0.5 (i.e., a player is equally likely to be a cheater or an honest player). For each time slice t, the inference carries out in two stages:

Stage 1: estimate the outcome of $\tilde{C_t}$ based on $\tilde{C_{t-1}}$, this estimation can be carried out by the following equation:

$$\begin{aligned} P(C_1 = T) \;=\; & P(C_t = T|C_{t-1} = T)P(C_{t-1} = T) \\ & +P(C_t = T|C_{t-1} = F)P(C_{t-1} = F) \qquad (1) \end{aligned}$$

Stage 2: updates $\tilde{C_t}$ with all of the evidences at time slice t. This computation can be carried out by the following equation:

$$\begin{aligned} & P(C_t = true|A_t, A_{t-1}, M_t^p, M_t^t, \triangle D_t, D_t) \\ & = \frac{P(a_t|a_{t-1}, C_t, m_t^p, m_t^t, \triangle d_t, d_t)P(C_t = true)}{\sum_{c=true}^{false} P(a_t|a_{t-1}, C_t = c, m_t^p, m_t^t, \triangle d_t, d_t)P(C_t = c)}. \end{aligned} \qquad (2)$$

IV. Experimental Evaluation

We have implemented three different aimbots for Cube. When the aimbot is enabled, it will find the nearest target and aim at it accurately. The aimbot will keep on aiming to the current target even there is a nearer target, until the distance between the player and the current target is larger than a certain threshold. If there is only one visible target, the aimbot will then keep on aiming this target until the target is invisible to the player.

The three aimbots perform similarly to the most common aimbots for first-person shooter games. The first one is the most popular and basic one, when enabled, it will aim at its target continuously. The second aimbot we built will automatically switch itself on and off for a random time interval which vary from 0.5 seconds to 2 seconds. The human player will temporary take over the control during the off periods. The justification for this feature is to reduce the aiming accuracy so that it is difficult to detect the cheater. The third aimbot we built is the most advanced one, it will create intensional misses for some random time intervals which vary from 0.5 seconds to 2 seconds. The aimbot pretends to miss like a human player by exhibits a smooth fluctuation of the crosshair around its target.

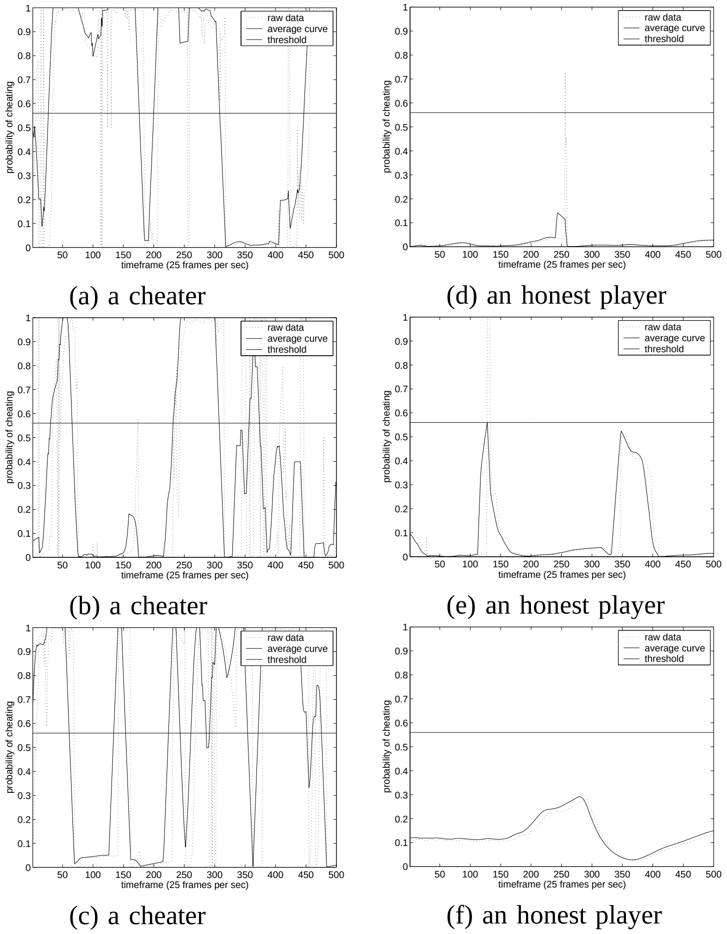
We carried out ten separate game sessions and then arrange the data into three data sets. These data sets are:

- Data set A, there are three honest players and three cheaters using the basic aimbot. Note that the data set A is used for training the dynamic Bayesian network.
- Data set B, there are three honest players and three cheaters using the basic aimbot.
- Data set C, there are three honest players, three cheaters using the auto-switching aimbot (the type II aimbot mentioned above) and three cheaters using the most advanced aimbot (type III aimbot mentioned above).

Experiment 1 – Effectiveness to Detect Cheating: In this experiment, we investigate the ability to detect cheaters while produce no false positive for honest players. We use the data set A as the training data and then infer the data set B. Figure 2 shows the probability of cheating over time for each of the six players in the data set B. Note that Figure 2(a)-(c) correspond to cheaters while (d)-(f) correspond to honest players. A predefined threshold is set to decide whether a player is a cheater or not.

The game sessions actually lasted for 10 minutes, but the figures are zoomed into the first 500 frames of the plays, i.e. 20 seconds, so that more details can be observed from the figures. For most of the time, the probability of an honest player keeps well below the threshold. On the other hand, the probability of a cheater can fluctuate above the threshold quite frequently, which indicates that the methodology is quite effective in detecting the use of aimbot.

There exists some time periods where a cheater is having a low probability of cheating, this probably occurs when the cheater does not have any visible target to aim at. It is common



(a) a cheater

(d) an honest player

(b) a cheater

(e) an honest player

(c) a cheater

(f) an honest player

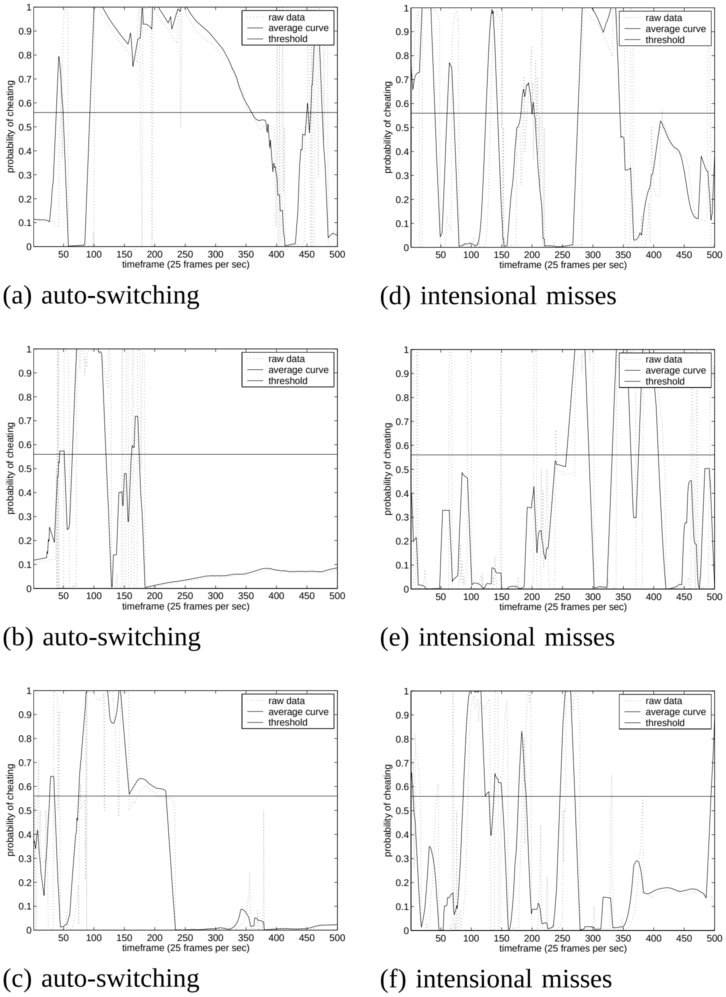
Fig. 2. Result of Experiment 1. Cheaters have probabilities fluctuating around the threshold, as illustrated in the sub-figures (a)-(c), while honest players have probabilities well below the threshold, as illustrated in sub-figures (d)-(f).

that only half of the time a player will have a visible target nearby, and this effect is magnify when the virtual world is a large one.

One may think that the detection can be improved by only counting the data when a player has any visible target. Unfortunately, for most of the first-person shooter game, the game server does not contain the information of static objects inside the virtual world, and this information is required to determine the visibility between any two players. Even if we include those information, the computation will be very expensive and thus it is not a scalable method.

Experiment 2 – Adaptiveness to Auto-switching and Intensional Misses: In this experiment, we investigate the ability to detect cheaters who use either one of the two more advanced aimbots, that is, they either perform auto-switching (turn on and turn off the aimbot alternately), or by intensionally missing some target. We still use the data set A as the training data and then infer the data set C. Figure 3 shows the probability of cheating over time for each player in data set C.

For the same reason that there is no visible target, there exists time periods that the probability drops far below the threshold. However, we look into the time frame from 100 to 350 in Figure 3a. During this period, the probability keeps beyond the threshold for 250 timeframes, or $250/25 = 10$ seconds, while the aimbot is switching on and off at about one second intervals. The probability does not drop even the aimbot is switched off, this is because the aimbot helps the player in aiming the target. We also look into the time frame from 250 to 400 in Figure 3d. During this period, the probability fluctuates around the threshold for 150 timeframes, or $150/25 = 6$ seconds, while the aimbot is creating intensional misses at about one second intervals. The probability drops when the aimbot misses its target, however, it rises again when the aimbot aims at its target in later time frames. This suggests that our methodology can effectively detect the use of aimbot even when the aimbot has the advanced feature to switching or missing intentionally.



(a) auto-switching

(d) intensional misses

(b) auto-switching

(e) intensional misses

(c) auto-switching

(f) intensional misses

Fig. 3. Result of Experiment 2. Probability of cheaters using an aimbot that automatically switching on and off occasionally in random intervals (a)-(c), and the probability of cheaters using an aimbot that creates intensional misses in random intervals (d)-(f).

Experiment 3 – Cross Validation: In this experiment, we validate our results by training and inferring with different combinations of data sets. We first train with data set B and infer with data set C, then train with data set B and infer with data set A, and so on. Figure 4 shows the probability of cheating over time for each test case. We use the same threshold for all of the test cases in determining whether the player is a cheater or not. From Figure 4, we see that even when we use different data set for the training, the methodology is still effective in determining whether a player is using the aimbot or not. The inferred probability of a cheater fluctuates above the threshold while the inferred probability of an honest player is below the threshold.

Scalability Analysis: We have carried out experiment on the scalability of our system. Due to the lack of space, we refer our readers to the technical report [6] for the experiment on investigating the scalability of our cheat detection framework.

V. Conclusions

Our work is the first attempt of using statistical inference in cheat detection. Experimental results show that the Dynamic Bayesian Network is an effective and scalable solution in the detection of the aiming robot cheat for a first person shooter multiplayer online game. Our framework only relies on the ordinary game states observed in the server side, therefore, cheaters cannot hack the detection system like hacking a cheat scanner software on the client side. The statistical approach has the advantage that one does not require to perform software update on the client side so as to detect new cheat, and the same methodology can be used to detect other kinds of cheat within the same category, because these cheats exhibit similar behavior (i.e., high aiming accuracy in all situations). We believe the proposed methodology and the prototype system provide a first step towards a systematic study of cheat detection and security research in the area of multiplayer online games.

References

[1] N. E. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. pages 104–113, 2001.

[2] B. D. Chen and M. Maheswaran. A cheat controlled protocol for centralized online multiplayer games. pages 139–143, 2004.

[3] Blizzard Entertainment. *Battle.net*. http://www.battle.net, 2002.

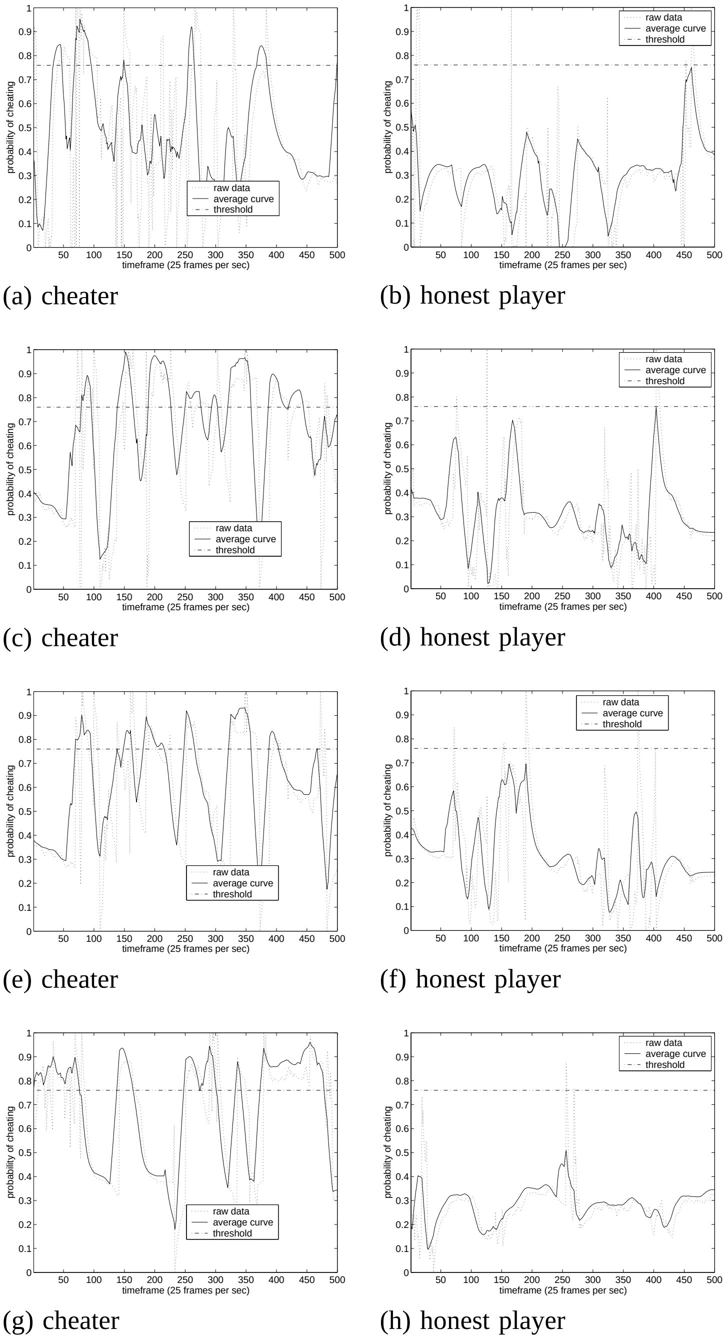
[4] Burton Filstrup Eric Cronin and Sugih Jamin. Cheat-proofing dead reckoned multiplayer games. 2003.

[5] Honghui Lu Oleg Sokolsky Usa Sammapun Insup Lee Christos Tsarouchis Margaret DeLap, Bjorn Knutsson. Is runtime vertification applicable to cheat detection. pages 134–138, 2004.

[6] Jiangchuan Liu Jeff Yan S.F. Yeung, John C.S. Lui. Detecting cheaters for multiplayer games: Theory, design and implementation. Technical report, The Chinese University of Hong Kong, 2005.

[7] Security Space. *Internet Research Reports*. http://www.securityspace.com/s_survey/data, 2004.

[8] Wouter van Oortmerssen. *Cube*. http://www.cubeengine.com, 2004.



(a) cheater

(b) honest player

(c) cheater

(d) honest player

(e) cheater

(f) honest player

(g) cheater

(h) honest player

Fig. 4. Result of Experiment 3. Use different combinations of data set for training and inference. Learn session B and infer session C: cheater (a) and honest player (b). Learn session B and infer session A: cheater (c) and honest player (d). Learn session C and infer session A: cheater (e) and honest player (f). Learn session C and infer session B: cheater (g) and honest player (h).