Editorial Manager(tm) for Multimedia Systems
Manuscript Draft

Manuscript Number: MMSJ-D-06-00047R2

Title: Dynamic Bayesian Approach for Detecting Cheats in Multi-Player Online Games

Article Type: Original Research Paper

Keywords: Multi-Player Game, Cheat Detection, Bayesian Network

Corresponding Author: Prof. John C. S. Lui,

Corresponding Author's Institution: The Chinese University of Hong Kong

First Author: Siu Fung Yeung

Order of Authors: Siu Fung Yeung; John C. S. Lui

Abstract: Massively multi-player games hold a huge market in the digital entertainment industry. Companies invest heavily in game developments since a successful online game can attract million of users, and this translates to a huge investment payoff. However, multi-player online games are 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 robots to play the game automatically and thus gain an unfair advantage. To overcome these problems, some popular online games release software patches constantly to block ``known'' hacks 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 or updates of the anti-cheating software 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 proposed method, we implement a prototype multi-player game system
to detect whether a player is using the ``aiming robot'' for cheating or not. Experiments show that we can effectively detect cheaters in a first-person shooter game with extremely low false positive rate. 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 multi-player games.

* Manuscript
Click here to download Manuscript: paper.tex

Multimedia Systems Journal:

Our Multimedia Systems Journal paper is an extension of our work ”Detect Cheating for Multi-Player Online Games: Theory, Design and Implementation” appeared in NIME 2006.

We have added the followings into the journal version.

1. Re-written Section 2 to add more related and background works.
2. Background knowledge on the dynamic Bayesian network are added in Section 3, we used a simplified version of our Bayesian model to illustrate the training process and the inference process.
3. A new pagragraph in Section 4.2 to discuss the motivation of our detection scheme.
4. A new pagragraph in Section 4.4 to discuss more about the overhead on the game server induced by the detection routine.
5. A new subsection “Multiple Thresholds” is added as Section 4.5. New ideas on how to obtain multiple thresholds and its usage are described.
6. A new pagragraph in Section 5.1 to describe the details of the players involved in the experiments.
7. More game states, increased from 6 to 8, are included in our Bayesian network for the cheat detection. New experiment results are obtained using the new model, and the results suggest that the new model is more effective in detecting cheaters.
8. In experiment 1 and experiment 2, new figures are included to show the aiming accuracy of each player against time to compare with the probability of cheating.
9. New cross-validate experiment in experiment 3. We carried out a cross-validate experiment on all of the datasets used in experiment 1 and experiment 2, the experiment shows the effectiveness in detecting cheaters while maintains a zero false positive rate.
10. New scalability experiment in experiment 4. The experiment shows the CPU usage on the game sever against varies number of connected players. The result shows that our system is scalable.
11. New Future Work section to discuss ideas on further improvement of our approach.

S.F. Yeung · John C.S. Lui

Dynamic Bayesian Approach for Detecting Cheats in Multi-Player Online Games

Abstract Massively multi-player games hold a huge market in the digital entertainment industry. Companies invest heavily in game developments since a successful online game can attract million of users, and this translates to a huge investment payoff. However, multi-player online games are 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 robots to play the game automatically and thus gain an unfair advantage. To overcome these problems, some popular online games release software patches constantly to block “known” hacks 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 or updates of the anti-cheating software 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 proposed method, we implement a prototype multi-player game system to detect whether a player is using the “aiming robot” for cheating or not. Experiments show that the proposed method can effectively detect cheaters on a first-person shooter game with extremely low false positive rate. 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 multi-player games.

S.F. Yeung
Department of Computer Science & Engineering, The Chinese University of Hong Kong
E-mail: sfyeung@cse.cuhk.edu.hk

John C.S. Lui
Department of Computer Science & Engineering, The Chinese University of Hong Kong
E-mail: cslui@cse.cuhk.edu.hk

1 Introduction

In 2004, multi-player online games generated billions of revenue. The Internet Research Report - Online Games 2004 [16] 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 September 2005, just the game “*World of Warcraft*” on its own has reached over $700 million revenue from monthly subscription fees with more than four million subscribers worldwide [12].

In the virtual world of a multi-player online game, players gain utility by interacting with different players around the world. Each player can directly control many objects (i.e., tanks, jeeps, planes, battleships, and even submarines,... etc) in the virtual world, and the actions will be periodically sent to the game server. In order to maintain view consistency, the game server needs to process the information and relays these updates to players within the same virtual world so that all players will have a synchronized view of the game [10]. Usually, each multi-player online game has its specific rules and regulations, for example, in order to play a game at a certain level, a player may choose to destroy n number of average moving avatars (which are graphical representation of other players in an online game), or he/she may opt to destroy m number of high moving avatars, where $n > m$. Strategizing and interacting with other players are the fun part of an online game. Players gain satisfaction by moving from a novice game level to an expert game level via destroying various avatars in the virtual world.

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 in the virtual world. As a matter of fact, cheating in multi-player online games is becoming so common because cheating software are easily accessible on the Internet. For example, Blizzard Entertainment once banned tens of thousand Diablo accounts whose players are believed to be cheaters [7].

Cheating in a first-person shooter game[1] 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 an unpleasant experience since an honest player will have very little chance, if any, in beating the cheater. 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 or the game console is in the hands of the cheaters, they can always work around any anti-cheating software. Besides using the software patches, online game companies also need to employ enough people to monitor the games 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 evidence[2] from other players of accusing someone for foul play, the administrators have the right to suspend a suspicious player from participating in the online game. Obviously, these types of solutions are labor intensive and they are not a scalable and effective solution for detecting cheaters for 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 (DBN) approach for our cheat detection framework. To test the effectiveness of the propose methodology, we also develop a prototype multi-player online 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 the detection engine runs in the server side only, therefore it is immune from hacks and it is a better deployable solution than the conventional software patches or human monitoring schemes. We evaluate our cheat detection method on a "*first-person shooter*" game for the detection of a specific type of cheat called aiming robot. We also carry out experiment on several enhancements of the aiming robot and the results show that our framework can detect the cheats effectively.

The remaining of this paper is organized as follows. In Section 2, related work on cheat detection or cheat prevention for multi-player online 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.

2 Background and Related Work

In this section, we provide some backgrounds on cheats and hacks for multi-player online games. We conclude this section by providing some related work in this area.

Multi-player online games divide into mainly four categories, i.e. turn-based game[3], massively multi-player online role-playing game (MMORPG), real-time strategy game (RTSG), and first-person shooter game (FPS). The types of cheat vary from one category to another category. In here, we survey some common types of cheat in each game category:

- *Turn-based Game:* **Look-ahead Cheat -** a form of cheat that let a player be the *last* player to make a decision on simultaneous actions. The cheater thus can make decision based on the decisions of other players.
- *MMORPG:* (1) **Trade Hack -** a form of cheat that attacks the security hole in network transaction. For example, some MMORPGs allow players to exchange items within the game, but the process may not be atomic such that when player A disconnects from the network during the exchange process, player A may be able to receive the item from player B but disconnects just before player B receives the item from player A. (2) **Item Grabber -** a form of cheat that automatically moves a player closer to the target item.
- *RTSG:* (1) **Map Hack -** a form of cheat that reveals hidden information of a game. The cheater will be able to reveal the whole map at the beginning of a game, which otherwise the exploration is a time consuming process. (2) **Speed Hack -** a form of cheat that change the timing in the client side. It will let the cheater to move and gather resources faster than other honest players.
- *FPS:* (1) **Wall Hack -** a form of cheat that modifies the display rendering of a game. The cheater will be able to see objects or other players behind opaque objects such as walls. (2) **Aiming robot-** a form of cheat that automate the movements of a player. It helps the cheater to aim accurately by moving the aiming point towards a nearby target. This type of cheat is very common in first-person shooter games.

There are research works on cheat-controlled protocols that prevent look-ahead cheats. In [5], authors proposed a

[1] A first-person shooter game is a type of game that each player will perceive from a first-person perspective view of the virtual world, the game concentrates mainly on aiming and shooting with various types of weapons and limited ammunition.

[2] Most online games have built-in support of game recording, one can replay the game from different viewpoints with the recorded file.

[3] A turn-based game is a game where each player plays in turn. A round of game will be over once every player has taken a turn, the result of that round will be processed and then all players will advance to the next round of play. Most board games (i.e., chess game) belong to this category.

lock-step protocol on a distributed game model wherein players will announce their decisions in cryptographically secure one-way hashes as commitments. Only when all players have announced their commitments, players then reveal their decisions 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 [6], authors proposed a scheme that enforces the fair-ordering of the message delivery so that cheating will be restricted to a certain level or may even be detected. In [8], author extended the idea so that the cheat-proof protocol can be used in games with dead reckoning. In [11], 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 this approach is not applicable to cheats that involve modification of client code that loads into memory at runtime. In [9], authors proposed the use of CAPTCHA tests [3] to ensure the participation of human players in online games. This approach is very effective against the use of fully automated robots. However, the insertion of the CAPTCHA tests can be quite annoying during the playing of an online game. And this approach is not applicable to cheats whose only assist the human player where a human player is still involved in the playing. In [14], the author briefly discussed the disadvantages of statistical approach to cheat detection in the game Quake. However, the detection scheme mentioned simply sort out players whose shoot too accurate, but didn't take other attributes or game states into account.

PunkBuster [4] is the first client-side cheat prevention system for commercial online games. To play an online game on a PunkBuster enabled server, the player must install the PunkBuster plug-in into the game client, otherwise the game server will refuse the connection. The PunkBuster client performs real-time memory scans to search for any known cheats and hacks. If any cheats or hacks are found, the PunkBuster client will report to the PunkBuster server, the cheater will be removed from the game and all other players will be informed of this violation. However, PunkBuster was once rendered nearly useless due to the very fast development cycle of the very famous cheat called OGC. OGC is a multifunctional cheat targets for several online FPS games including the very famous and popular commercial FPS game called Counter-Strike [18]. Representatives from PunkBuster asked for financial and development support from Counter-Strike's developer, Valve, but were turned down. Thus the involvement of PunkBuster with Counter-Strike was over. Currently, PunkBuster is supported on 18 online FPS games including the well-known Quake series.

After Valve decided to forego PunkBuster, Valve developed its own anti-cheat system called Valve Anti-Cheat (VAC) [19]. All game clients and game servers developed by Valve run VAC by default. Valve keeps much of VAC's detection method secret, but it is believed that VAC uses hash detection to identify individual cheats. Therefore, new cheats must be discovered and added to the VAC database. It is also possible that VAC detects processes that work like cheats, such as processes that hook on the game client at run-time. This once caused some problems such as false positive would be produced if a player runs the in-game MP3 player called HLamp.

HLGuard [15], formerly called CSGuard, is a free server-side anti-cheat system for the famous commercial FPS game, Half-Life, and its many variations such as Counter-Strike. HLGuard contains a set of anti-cheat features and includes a wall-hack blocking function and an aiming-robot detection routine. However, HLGuard uses a very simple algorithm in its aiming-robot detection routine. Basically, it looks for players whose kill other players in the way like an aiming robot does, such as changes the aiming direction dramatically within a very short period of time and then stops at aiming a target accurately [1] and [2]. The algorithm is too simple that many subsequent aiming robots could not be detected by HLGuard properly. For this reason, the detection routine has been disabled by default in the newer releases of the HLGuard.

3 Bayesian Networks

In this section, we provide a brief introduction of Bayesian network and its application in detect online game cheating.

A Bayesian network can be viewed as a graphical representation of causal relationships between different events. In general, a Bayesian network is an acyclic graph that consists of a set of nodes and a set of directed edges. Each node in a Bayesian network represents a random variable (i.e., which can have discrete or continuous outcomes), and each directed edge represents a causal influence from a parent node X_p to its child X_c. A node can have more than one parent, it implies that the random variable is *dependent* on many random variables (represented by all the parent nodes). A node can be parent of more than one node, in this case, this implies that the random variable can influence many other possible random variables (all its children nodes). We define the set of parent nodes of X as $parents(X)$. The probability of a node X is conditioned on only its parent nodes and conditionally independent on other random variables. We say that $P(A|B,C) = P(A|B)$ if $B \in parents(A)$ while $C \notin parents(A)$. Thus, the joint probability distribution of a Bayesian network is given by:

$$\begin{aligned} & P(X_1, X_2, ..., X_n) \\ = \; & P(X_1|parents(X_1))P(X_2|parents(X_2)) \\ & \ldots \quad P(X_n|parents(X_n)). \end{aligned}$$

To illustrate the applicability of this joint probability distribution, let us consider an example in Figure 1(a), which illustrates a Bayesian network that models the causal relationship between three random variables: $Cheating(\tilde{C})$, $Distance(\tilde{D})$ and aiming $Accuracy(\tilde{A})$. In here, the random variables $Cheating$ and $Distance$ are parent nodes of $Accuracy$. Therefore, whether the player is a cheater or not and his distance from the aiming target can influence the outcome of the aiming $Accuracy$. In this case, the probability of the variable $Accuracy$ is dependent upon the values of $Cheating$ and $Distance$, while the variables $Cheating$ and $Distance$ themselves are independent of any random variable. The joint probability distribution of this Bayesian network is given by:

$$\begin{aligned}&P(\tilde{C},\tilde{D},\tilde{A})\\&= P(\tilde{A}|parents(\tilde{A}))P(\tilde{C}|parents(\tilde{C}))P(\tilde{D}|parents(\tilde{D}))\\&= P(\tilde{A}|\tilde{C},\tilde{D})P(\tilde{C})P(\tilde{D}).\end{aligned}$$

3.1 Training and Inferring Bayesian Network

In some situations, the values of both random variables $Accuracy$ and $Distance$ are observable and one may want to determine the probability of the variable $Cheating$ given the known values, i.e., $P(\tilde{C}|\tilde{A},\tilde{D})$. This can be done by inference. Inferring a Bayesian network is the task to compute the probability of a node given the values of all other nodes. But before one can infer the Bayesian network, one must know the *prior probabilities* of all nodes in the network, these probabilities can be obtained via the training process.

$Cheating(\tilde{C})$	$Distance(\tilde{D})$	$Accuracy(\tilde{A})$
$True$	1	1
$True$	1	1
$True$	0	1
$True$	0	0
$False$	1	1
$False$	1	0
$False$	0	1
$False$	0	0
$False$	0	0
$False$	0	0

Table 1 Data set obtained in the training stage for the Bayesian network in Figure 1(a). (In real cases the values of $\tilde{D}$ and $\tilde{A}$ should be continuous values, this example uses very simple dataset for easier understanding on the idea.)

Training the network is a task of sampling the values of the variables for a set of test cases. For example, let $Cheating$ be a discrete random variable which has two possible outcomes, true or false, i.e., $\tilde{C} \in [true, false]$. The variable aiming $Accuracy$ is a discrete random variable with outcome of either 1 (accurate) or 0 (inaccurate), i.e. $\tilde{A} \in [0,1]$. Finally, $Distance$ is also a discrete random variable which can have the value of either 0 (i.e. very near to the target) or 1 (i.e. far away from the target), i.e. $\tilde{D} \in [0,1]$. If we record the values of $Accuracy$, $Cheating$ and $Distance$ of a particular player at a game session, we may obtain a set of data such as the one shows in Table 1. Each row of the table corresponds to a single sampled data. If we count the "frequency" of each possible value of the variables and normalize the frequencies to values between zero and one, we will get the following probability distributions:

$$P(\tilde{C}) = \begin{cases} 0.4\ \tilde{C} = true \\ 0.6\ \tilde{C} = false, \end{cases}$$

$$P(\tilde{A}|\tilde{C},\tilde{D}) = \begin{cases} 0.25\ \tilde{A}=1, \tilde{C}=false, \tilde{D}=0 \\ 0.75\ \tilde{A}=0, \tilde{C}=false, \tilde{D}=0 \\ 0.50\ \tilde{A}=1, \tilde{C}=false, \tilde{D}=1 \\ 0.50\ \tilde{A}=0, \tilde{C}=false, \tilde{D}=1 \\ 0.50\ \tilde{A}=1, \tilde{C}=true, \tilde{D}=0 \\ 0.50\ \tilde{A}=0, \tilde{C}=true, \tilde{D}=0 \\ 1.00\ \tilde{A}=1, \tilde{C}=true, \tilde{D}=1 \\ 0.00\ \tilde{A}=0, \tilde{C}=true, \tilde{D}=1 \end{cases}$$

With the prior probabilities, we can now infer the Bayesian network. Assume at a particular instance, we observed that the shooting is accurate (i.e., $\tilde{A} = 1$) and the player is far away from the target (i.e., $\tilde{D} = 1$). Now we can infer the probability of whether the player is cheating or not at that instance. By Bayes' rule, the probability is given by:

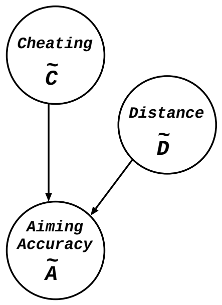
$$\begin{aligned}&P(\tilde{C}=T|\tilde{A}=1,\tilde{D}=1)\\&=\frac{P(\tilde{A}=1|\tilde{C}=T,\tilde{D}=1)P(\tilde{C}=T)P(\tilde{D}=1)}{\sum_{\tilde{C}} P(\tilde{A}=1|C,\tilde{D}=1)P(C)P(\tilde{D}=1)}\\&=\frac{P(\tilde{A}=1|\tilde{C}=T,\tilde{D}=1)P(\tilde{C}=T)}{P(\tilde{A}=1|\tilde{C}=T,\tilde{D}=1)P(\tilde{C}=T)+P(\tilde{A}=1|\tilde{C}=F,\tilde{D}=1)P(\tilde{C}=F)}\\&=\frac{(0.5)(0.4)}{(0.5)(0.4)+(0.5)(0.6)}\\&=0.4.\end{aligned}$$

Therefore, at any particular instance during a game, the Bayesian network approach allows us to determine the probability distribution of $\tilde{C}$. That is, whether the player is cheating or not, bases on the observed values of other random variables.

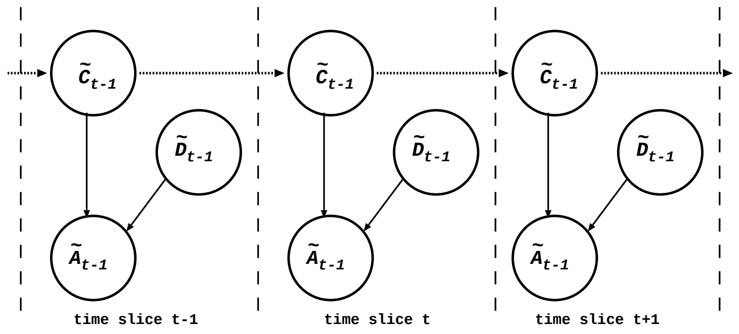
3.2 Dynamic Bayesian Networks and Inferring Information

Although the BN approach can help us to estimate the probability of whether the player is cheating or not, but the approach has a *major limitation* since it can only model the static behavior of a player. Note that the outcomes of the random variables $\tilde{C}$, $\tilde{A}$ and $\tilde{D}$ can be time dependent, therefore, a temporal representation is needed so that one can accurately predict whether the player is a cheater or not.

When random variables take on different values as time elapses while the structure of the inference network remains unchanged, the resulting temporal model is called a *Dynamic Bayesian Network* (DBN). Typically, a DBN is composed by replicating the same Bayesian network for different time intervals or time slices. Dependencies not only exist between nodes inside the same time slice, but a node within one time slice may also be dependent on the same node and/or other nodes in previous time slices. To illustrate, consider the example in Figure 1(b) which illustrates a DBN formed by



(a) A Bayesian network consists of three random variables aiming $Accuracy$, $Cheating$ and $Distance$.



(b) A DBN unrolled for three consecutive time slices.

Fig. 1 An example of Bayesian network and Dynamic Bayesian network.

replicating the Bayesian network in Figure 1(a) for every time slices. This enhancement adds a dependency between the random variable $Cheating(\tilde{C})$ in every two adjacent time slices. In other words, the random variable $\tilde{C}_t$ at time t depends on the random variable $\tilde{C}_{t-1}$ at time $(t–1)$.

Inferring a DBN is much like inferring a static Bayesian network, but one will not infer the whole dynamic Bayesian network since the state space is large (or infinite) and it is computational expensive to deal with an arbitrary large dynamic Bayesian network as time elapses. Instead, one needs to update the probabilities and "roll" the network forward as time elapses. We illustrate this concept with a working example. Figure 1(b) models the temporal change in the probability of whether the player is cheating or not, as well as the influence of the random variable $\tilde{C}_t$ and $\tilde{D}_t$ on the probability of the player's aiming accuracy.

Without the loss of generality, the very first time slice occurs at $t = 0$ is considered as a special case with the assumption that the prior probability $P(\tilde{C}_0 = true) = 0.5$ (i.e., in the beginning, we have no information to determinate whether a player is a cheater or not. So we consider the case that a player can be honest or cheating with equal probabilities). Let say that in the first time slice, we observe that the player aims accurately from a short distance, i.e., $\tilde{A}_1 = 1$ and $\tilde{D}_1 = 0$. For the second time slice, we also observe that the player aims accurately from a long distance, i.e., $\tilde{A}_2 = 1$ and $\tilde{D}_2 = 1$. Given the outcomes of $\tilde{A}_t$ and $\tilde{D}_t$, for $t \in \{1, 2\}$ are known, one can infer the probability of whether the player is cheating or not on each of these two time slices.

The inference can be carried out in *two stages*. For the first stage, we estimate whether $\tilde{C}_t$ is true or not based on the probability distribution of $\tilde{C}_{t-1}$. For the second stage, we improve the estimate of $\tilde{C}_t$ based on the observed values of $\tilde{A}_t$ and $\tilde{D}_t$ at time t. Let us illustrate this concept.

On the first time slice, estimate the outcome of $\tilde{C}_1$ based on $\tilde{C}_0$. Note that this estimation is computed based on the conditional probability distribution $P(C_t|C_{t-1})$. The estimate of $\tilde{C}_t$ is:

$$\begin{aligned}
&P(\tilde{C}_1 = T)\\
&= P(\tilde{C}_1 = T|\tilde{C}_0 = T)P(\tilde{C}_0 = T)\\
&\qquad + P(\tilde{C}_1 = T|\tilde{C}_0 = F)P(\tilde{C}_0 = F)\\
&= (0.8)(0.5) + (0.3)(0.5)\\
&= 0.55.
\end{aligned}$$

For the second stage, we update this estimate base on the observed values of $Accuracy$ and $Distance$ (i.e., $\tilde{A}_1$ and $\tilde{D}_1$) for time slice $t = 1$, which gives:

$$\begin{aligned}
&P(\tilde{C}_1 = T|\tilde{A}_1 = 1, \tilde{D}_1 = 0)\\
&= \frac{P(\tilde{A}_1 = 1|\tilde{C}_1 = T, \tilde{D}_1 = 0)P(\tilde{C}_1 = T)P(\tilde{D}_1 = 0)}{\sum_{c_1} P(\tilde{A}_1 = 1|\tilde{C}_1 = c_1, \tilde{D}_1 = 0)P(\tilde{C}_1 = c_1)P(\tilde{D}_1 = 0)}\\
&= \frac{(0.5)(0.55)}{(0.5)(0.55) + (0.5)(0.45)}\\
&= 0.38.
\end{aligned}$$

This gives the probability of whether the player is cheating or not at time slice $t = 1$.

For the second time slice, we estimate the outcome of $\tilde{C}_2$ based on updated probability of $\tilde{C}_1$, this gives:

$$\begin{aligned}
&P(\tilde{C}_2 = T)\\
&= P(\tilde{C}_2 = T|\tilde{C}_1 = T)P(\tilde{C}_1 = T)\\
&\qquad + P(\tilde{C}_2 = T|\tilde{C}_1 = F)P(\tilde{C}_1 = F)\\
&= (0.8)(0.38) + (0.3)(0.62)\\
&= 0.49.
\end{aligned}$$

Then we update this estimate based on the observed values of $\tilde{A}_2$ and $\tilde{D}_2$, this gives:

$$\begin{aligned}
&P(\tilde{C}_2 = T|\tilde{A}_2 = 1, \tilde{D}_2 = 1)\\
&= \frac{P(\tilde{A}_2 = 1|\tilde{C}_2 = T, \tilde{D}_2 = 1)P(\tilde{C}_2 = T)P(\tilde{D}_2 = 1)}{\sum_{c_2} P(\tilde{A}_2 = 1|\tilde{C}_2 = c_2, \tilde{D}_2 = 1)P(\tilde{C}_2 = c_2)P(\tilde{D}_2 = 1)}\\
&= \frac{(1.0)(0.49)}{(1.0)(0.49) + (0.5)(0.51)}\\
&= 0.66.
\end{aligned}$$

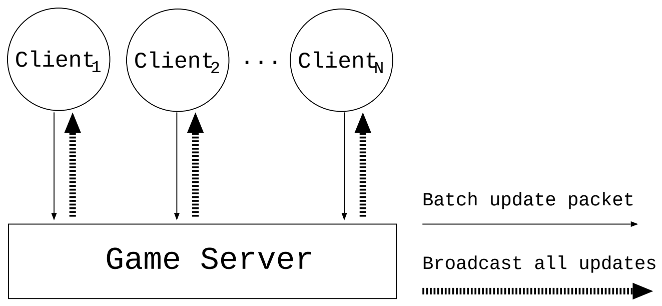


Fig. 3 Updates between playing clients and the game server in Cube.

Intuitively, the probability that the player is cheating in the current time slice increases if the player cheated in the previous time slice. This probability is also reinforced by the observed evidence that the player can keep a high aiming accuracy even when he moved away from the target (i.e., $\tilde{D}_1 = 0$ and $\tilde{D}_2 = 1$).

4 Design and Implementation

In here, we present our framework of cheating detection with a multi-player online game engine. To demonstrate the effectiveness of the proposed method, we built a prototype of a multi-player game system using the open source first-person shooter game Cube [13] 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 an unfair advantage. To realize our dynamic detection scheme, we also implemented a dynamic Bayesian inference routine on the game server so that we can infer the probability of cheating for each individual players during a game session. We implemented three of the most common aimbot variants, and evaluate the adaptiveness of the proposed detection framework towards these different aimbots. Moreover, we measure the overhead of the detection framework on the game server so that we can evaluate the *scalability* of the detection framework.

4.1 Multi-Player Game Architecture

In a real-time multi-player online game, all of the players must maintain a consistent view of the game states. The perception of the real-time interaction is achieved by updating the game states among all players frequently. Typically, state updates will be carried out at a fixed time interval called a timeframe. This is to avoid overwhelming the network bandwidth and the processing power of the server. Updates on multiple game states within the same timeframe will be *batched* and sent out as a single update packet, and the typical length of a timeframe is in tens of millisecond. Figure 2 illustrates the updates sent from and received by the clients over time with a timeframe of 50 ms.

We used the game Cube as our testbed. Cube employs client-server architecture. Each game client sends its update to the game server at every timeframe. When the server receives these updates from different clients, the game server will then broadcast these updates to all players in the virtual world so every player can maintain the view consistency. This type of communication mode is illustrated in Figure 3. Cube clients synchronize at a timeframe of 40ms, or 25 frames per second. Each update from a client includes the following information (i.e. game states): 1) the position of the player, 2) the direction and velocity of the player, 3) the aiming direction of the player and, 4) the action of the player, e.g. strafing, jumping, . . ., etc.

Our cheating detection framework will only rely on these game states. Therefore, the detection framework can reside completely on the server side and it does not require any modification of the client code. This simplifies the deployment process since one can easily enhance the detection framework transparently to the players if a new form of cheating is suspected.

4.2 Aimbot

Aimbot is a very popular type of cheat used in first-person shooter games. When this form of cheat is enabled, it will take over the control of the player's mouse pointer and locks the player's aiming point, which is called "*crosshair*", onto a nearby target. To illustrate the effect on using an aimbot, Figure 4(a) shows a scenario when the aimbot is enabled and the cheater aims at a target while the cheater is moving in the virtual world. One can notice that the crosshair is *always* locked onto the aiming target. In contrast, Figure 4(b) shows a similar scenario but the player is not using any aimbot cheat. In this case, one can observe that the crosshair of the honest player may not be able to track the target all the time (i.e., the last three frames in the figure). Based on this scenario, it is easy to observe that when an aimbot is enabled, the cheater can easily destroy any target and gain an unfair advantage over other honest players. In general, a cheater using an aimbot will exhibit very outstanding accuracy in aiming and very low, if any, miss rate in any situations.

To avoid being detected due to its high aiming accuracy, some advanced aimbots may pretend to act as a normal player, either by automatically switches itself on and off periodically, or by creating some intended misses from time to time. However, human players have diverse behaviors but cheaters using aimbots are very likely to exhibit some kind of patterns in their cheating behaviors. For example, a skillful player who is good at shooting will adopt a specific tactic to gain the highest advantage according to the current circumstance and landform of the virtual world. However, a cheater may exhibit outstanding accuracy but lack the sense of paying attention to the environment of the virtual world. An experienced administrator can tell whether a player is skillful or cheating by carefully reviews the recorded game. In most cases the cheaters would configure the aimbots such that they would perform much better than the cheaters them-

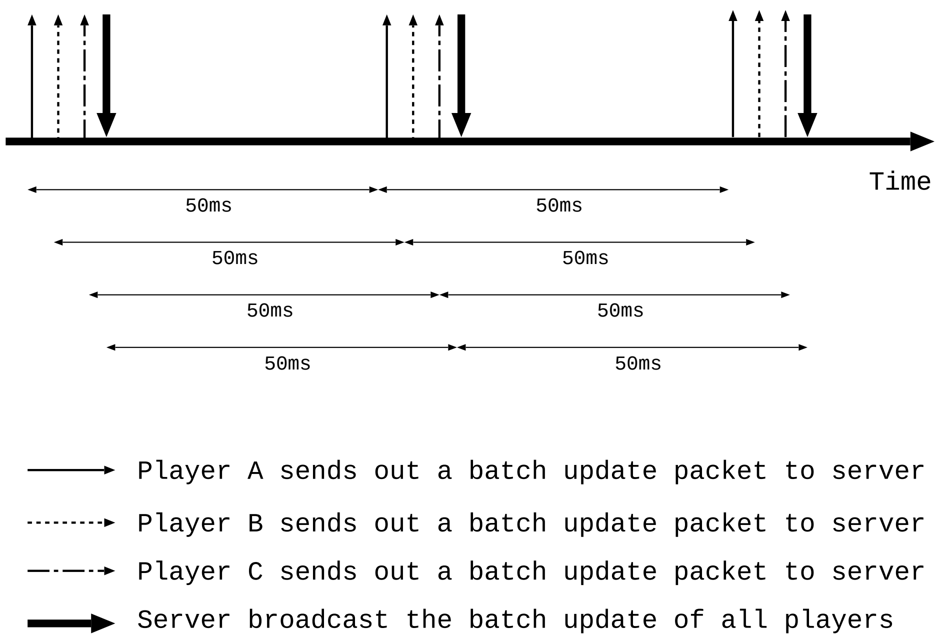

Time
50ms
50ms
50ms
50ms
50ms
50ms
50ms
50ms
Player A sends out a batch update packet to server
Player B sends out a batch update packet to server
Player C sends out a batch update packet to server
Server broadcast the batch update of all players

Fig. 2 Network architecture of multi-player online game: updates sent from and received by clients over time with a timeframe of 50 ms.

selves when they are unassisted. However, if a cheater configures the aimbot to make only a slight improvement in performance, e.g., by automatically turning on occasionally or even rarely, there are still clues that one can observe the presences of an aimbot at the very moment when the aimbot is activating. For example, the most advanced aimbot today is called the "charged-type" aimbot, these aimbots are turned off by default but will only activate when the cheater presses a self-defined hotkey, and then turned off immediately once the cheater releases the hotkey. More advanced aimbots would only activate when the crosshair is sufficiently close to the target, makes a more natural look therefore harder to detect. These aimbots are completely unnoticeable in peacetime but may still be caught when they are activated, although very careful and professional reviews are required. These are the reasons why the most authoritative cheat detection method used today is still by human observation, and it is especially important in major tournaments [1] and [17].

Note that above are observations we made in the FPS type of game. Based on these observations, we are able to make the necessary Bayesian model to detect aimbot. We believe the proposed approach holds promise in other forms of games or cheats, although one may need to make some specific observations and derive new Bayesian models for different types of games.

4.3 **Our Dynamic Bayesian Network**

The overview of the DBN we used for our experiments is illustrated in Figure 5. 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.

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 compares 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 moving the crosshair or not, which is denoted by the random variable P, and
5. the distance between the player and the aiming target, which is denoted by the random variable D.
6. the difference in D between this time slice and previous time slice, which is denoted by the random variable $\triangle D$.
7. the difference in A between this time slice and previous time slice, which is denoted by the random variable $\triangle A$.

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 landforms and location of 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

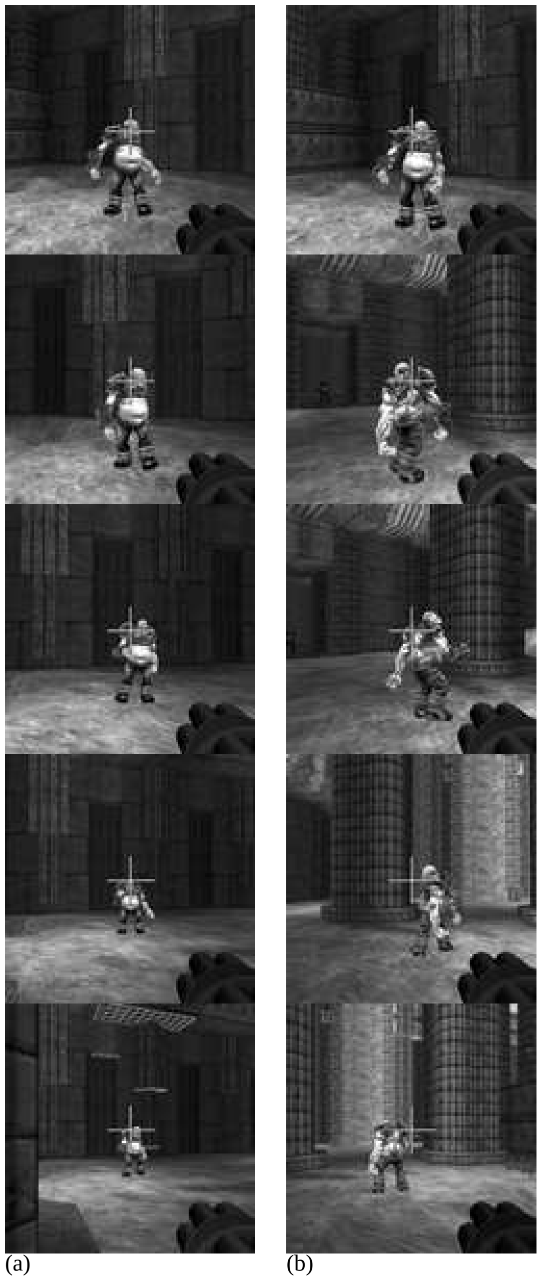


Fig. 4 (Left) consecutive frames when a cheater using an aimbot aiming at a target. The '+' mark at the center is the aiming point called 'crosshair'. Notice that the crosshair aims the target accurately even the player is moving during these five frames. (Right) consecutive frames when an honest player aiming at a target. The player trying to aim the target but is not accurate since the player is moving during these five frames.

above parameters themselves as independent random variables. Moreover, we model the aiming process as a first order Markov process. Because aiming is a fine tuning 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 modeled 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. This allows us to capture the feature of an advanced aimbot which we mentioned previously.

4.4 **Training and Inference**

All of the random variables we need to infer the probability of cheating, i.e., C, M^p, M^t, P, D, $\triangle D$, $\triangle A$, can be obtained directly, or derived from the game states contained in the update packets by various players. For example, by comparing the current position and the latest recorded position of a player, we know whether the player is moving or not. During a game session, a player will meet different targets in the virtual world, and it is very common to meet more than one target at the same time. We define a player's current target as the latest target the player's crosshair hovered over. To illustrate, let player A's crosshair hovered over player B, then for player A, $\tilde{Target_A} = B$ until player A's crosshair hovered over another player or player B became invisible to player A. The variables Distance (D) and Accuracy (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 a finite field. 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. Moving crosshair ($\triangle P$), with $\triangle P \in [true, false]$,
5. Distance from the aiming target (D), with $D \in [0, 1, 2, 3]$ in which larger the value implies further away is the distance,
6. Change in D ($\triangle D$), with $\triangle D \in [0, 1, 2, 3]$,
7. Aiming accuracy (A) with $A \in [0, 1, 2, 3]$ in which the lower the value implies higher is the aiming accuracy and,
8. Change in A ($\triangle A$) with $\triangle A \in [0, 1, 2, 3]$.

Using these data, one can compute the following prior probability distributions:

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

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

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

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:

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

The pseudo-code to perform this inference computation on the game server is as follows:

Server Loop

```
1.  while (gameover == false) {
2.      for each player() {
3.          playerStates = receive_player_state_update();
            // stage 1 computation based on Eq. (1)
4.          pPredict = cptCheating[cheating] * pCheating[t-1]
              + cptCheating[!cheating] * (1.0 - pCheating[t-1]);
            // stage 2 computation based on Eq. (2)
5.          T = cptAccuracy[playerStates][cheating]
              * pPredict;
6.          F = cptAccuracy[playerStates][!cheating]
              * (1.0 - pPredict);
7.          pCheating[t] = T / (T + F);
8.      }
9.      timeframe++;
10. }
```

Computational Analysis: Based on the above pseudo-code, we see that to estimate the cheating probability, we require a total of nine floating point operations (from line 4 to line 8) per player within each time slice. One can easily see that this is a light weighted computation. Assuming that each time slice is 40 ms, i.e. 25 frames per second, this implies that to detect 100,000 simultaneous players, it only requires $9 * 25 * 100,000 = 22,500,000$ floating point operations per second. Using a P4 2.0GHz server as an example, which can perform about one billion floating point operations per second, therefore the overhead to detect 100,000 simultaneous players uses only 22.5% of the CPU capacity.

For example, in commercial games such as counter-strike and Quake, the game clients send all commands (keyboard inputs) to the server. The server runs the simulation of the whole game because all game states must be authorized by the server before the server broadcasts the information to other players. Therefore, the server already knows the position and the aiming direction of all players at each instance of the game. In our proposed method, the only additional computation requires to perform prior to the Bayesian inference routine is to compute the "distance between a player's aiming trajectory and the player's current target", i.e., the player's accuracy. That is, to compute the distance between a point and a line which involves a cross-product operation. Moreover, as we mentioned before, the computation only occurs when there is any visible target, and the server only needs to trace the accuracy with respect to the current target, until it is not visible to the player or their distance is beyond a certain threshold. Also, for FPS games the number of players involved in the same game session typically varies from 8 to 32. Hence, the computational overhead induced should not be much. Moreover, in practical implementation one could create an individual queue to store a copy of the user states, and then carries out the computation of the player's accuracy and also the Bayesian inference routine in a computational thread, which is separated from the main server loop so as to reduce the real-time workload on the server.

4.5 Multiple Thresholds

Using a higher threshold value can ensure zero false positive rates, however, the system will then be less sensitive to cheaters. On the other hand, using a lower threshold value can filter out most potential cheaters but false positives could be annoying to the players and server administrators. Hence, instead of using a single threshold, it may be more practical to have multiple thresholds in real-life situations. In the training session, we could divide players into different groups according to their playing skills, such as grouping them into average players, advanced players and experts. Then, for each of the groups, we obtain a separate threshold value by training the Bayesian Network with the corresponding data set. The more skillful the players in the group, the higher the obtained threshold value should be.

In real-life situations, players often join ongoing game sessions in public servers without knowing who are the other players, it is very common that a mixture of players with different skill levels playing together. A server administrator may like to configure the system such that different alerts will be triggered when a player's inferred probability excesses different thresholds. For example, when the probability excesses the first threshold, the player will be logged but no action will be taken. However, when the probability excesses the second threshold, an alarm will be sent to the administrator and gameplay recording will be enforced. If the last threshold has been reached, the player would be flagged as cheater and be removed from the game immediately.

5 Results

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 at 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 similar to the most common aimbots of first-person shooter games. They are:

– **Type I:** The first one is the most popular and basic one, when enabled, it will aim at its target continuously.
– **Type II:** 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 be detected as a cheater.
– **Type III:** The third aimbot we built is the most advanced one, it will create intentional 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 separated game sessions. There are a total of sixteen players. The players include eight student volunteers and eight gamemates of the authors. They range from naive players to above-average players, most are average level players. We arranged the data into three data sets. These data sets are:

– Data set A, there are three honest players (i.e. h_{a1}, h_{a2}, h_{a3}) and three cheaters (i.e. c_{a1}, c_{a2}, c_{a3}) using the basic aimbot (the type I aimbot mentioned above). Note that the data set A is used for training the DBN.
– Data set B, there are three honest players (i.e. h_{b1}, h_{b2}, h_{b3}) and three cheaters (i.e. c_{b1}, c_{b2}, c_{b3}) using the basic aimbot.
– Data set C, there are three honest players (i.e. h_{c1}, h_{c2}, h_{c3}), three cheaters (i.e. c_{c1}, c_{c2}, c_{c3}) using the auto-switching aimbot (the type II aimbot mentioned above) and three cheaters (i.e. c_{a4}, c_{a5}, c_{a6}) using the most advanced aimbot (type III aimbot mentioned above).

During session A, no inference is carried out, but the data collected are used for the training of our Bayesian Network. We have modified the game client such that each time when an aimbot is activated or deactivated, it is logged along with other game states. We then use data set A to compute the two prior probabilities mentioned in Section 4.4, i.e.

1. $P(C_t|C_{t-1})$, and
2. $P(A_t|A_{t-1}, C_t, M^p_t, M^t_t, P_t, D_t, \triangle D_t, \triangle A_t)$,

During game session B and C, inference takes place in real time. At each time frame, the system updates the probability of cheating for each of the players. Note that we also logged the states of each player into files, which can later be used for offline training or offline inference. Hence, we can also train and infer the data with different combinations (cross validation) in an offline manner.

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 6 shows the probability of cheating over time for each of the six players in the data set B. Note that Figure 6(a)-(c) correspond to cheaters while (d)-(f) correspond to honest players. The threshold line in each sub-figure corresponds to the highest probability ever reached by an honest player, which is obtained in the training section.

Actually, the game sessions each 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 that only half of the time a player will have a visible target nearby, and this effect is magnified 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 this information, the computation will be very expensive and thus it is not a scalable method.

Experiment 2 - Adaptiveness to Auto-switching and Intentional 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 (turns on and off itself alternately), or by intentionally missing the current target. We still use the data set A as the training data and then infer the data set C. Figure 7 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 350 to 450 in Figure 7(a). During this period, the probability keeps beyond the threshold for 100 timeframes, or $100/25 = 4$ seconds, while the aimbot is switching on and off periodically at random intervals. The probability does not drop even the aimbot is switched off, this is because aiming the target becomes much easier with the periodic assistance from the aimbot.

We also look into the time frame from 0 to 150 in Figure 7(d). During this period, the probability fluctuates around the threshold for 150 timeframes, or $150/25 = 6$ seconds, while the aimbot is creating intentional 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. The results suggest that our methodology can effectively detect the use of an aimbot even if the aimbot has the advanced feature to switch automatically and miss intentionally.

The results also suggest that our methodology actually learns and detects “how do aimbots aim”. For example, at around the 250th timeframe in Figure 7(k), the aiming accuracy rises from zero to nearly 100% and then drops back to zero in a period of around 25 timeframes (i.e. 1 second). It indicates that the aimbot is in action and at the same time a visible target exists. Referring to the corresponding inferred result in Figure 7(e), the probability of cheating rises sharply at the same moment when the aiming accuracy rises. Therefore, even though the aimbot is only activated for about 25 timeframes, it is being detected shortly after it aimed at a target accurately. ¿From other results, we can also observe that many times the probability rises above the threshold shortly after the aiming accuracy goes high. It is because ”aims make by aimbots” are being detected. For the results corresponding to the honest players, there are also sometimes the aiming accuracy rises sharply such as near the 400th timeframe in Figure 6(k). However, the probability of cheating still keeps below the threshold. This is because not only the aiming accuracy, but also other game states, influences the inference result.

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 data set C, then train with data set B and infer data set A, and so on. Figure 8 shows the probability of cheating over time for each test case. We use the same threshold for all the test cases in determining whether the player is a cheater or not. ¿From Figure 8, we see that even when we use different data set as 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 keeps below the threshold.

Experiment 4 - Scalability of using the Inference Engine: In this experiment, we investigate the overhead of the dynamic Bayesian inference routine on the game server. We measure the maximum CPU load on the game server from one player to ten players. We first carry out the measurement on an original Cube server. We then repeat the measurement on the Cube server with an added dynamic Bayesian inference routine. Figure 9 shows the percentage of CPU usage against the number of connected players on the Cube server running on a P4 2.4GHz machine with 512MB memory. Although the CPU loading of the original Cube server is not linear to the number of connected clients, this is probably because the Cube server is not optimized in its performance, as it is mentioned in the Cube’s documentation. However, compares the modified server running the Bayesian inference routine to the original server, there is only a constant factor induced by the Bayesian inference routine. This suggests that the inference algorithm is scalable on massive multiple player online games where the number of connected players per server is of the order of thousands.

6 Future Work

The proposed approach in this paper is applied on a FPS game, although it may be specific to this type of game, we believe the methodology holds promise and open doors for new detection method for other types of games. For example, based on the Bayesian network used in the Section 5, we may be able to further enhanced the effectiveness of our scheme on FPS games by adding more nodes into the Bayesian

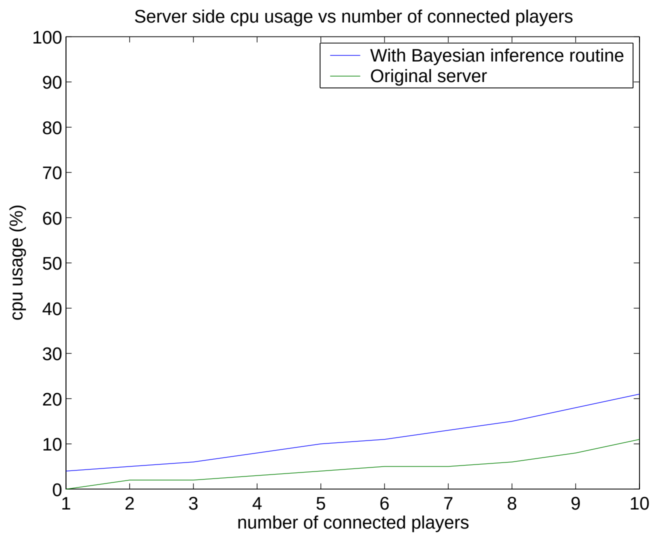


Fig. 9 % of CPU loading on a Cube server running on P4 2.4GHz machine with 512MB memory.

network, for example:

1. weapon - the player's current weapon,
2. map type - discrete values represent close up battle, long range battle or in-between,
3. number of visible targets - the number of targets that currently visible to the player,
4. action - the player's current action, such as dodging, jumping, standing, walking or running,
5. skill - a value indicate how good the player should be, discuss below.

Since the inspiration of our approach is to imitate how professional administrators detect cheaters by observation. The motive of adding these new nodes is trying to capture the skill of the player and capture more information of the whole game. The data needed to the proposed "skill" node could be obtained by a centralized method. For example, many popular FPS games have regular national or international tournaments such as the Cyberathlete Amateur League (CAL) tournaments. One of the most critical issue is to differentiate CAL-level players from cheaters, which means false positive should not happen even for very skillful honest players. Since nowadays most popular FPS games use global login ID to identify players, which can only be obtained by purchasing a new copy of the game, and there exists centralized global databases for all servers to retrieve and save banned login ID. Therefore, it is possible to query a player's tournament history according to the global login ID and determine the player's "skill" accordingly.

7 Conclusions

Our work is the first attempt of using statistical inference in cheat detection. Different from HLGuard, our approach makes use of machine learning to capture the behaviors of different aimbots. Moreover, our Bayesian model includes a variety of available game states so that the variations in the player's performance could also be taken into account in the inference. Our 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 multi-player online game. Our framework only relies on the 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 updates on the client side so as to detect new released cheats, and the same methodology can be used to detect other cheats of the same category, because these cheats exhibit similar behavior (e.g., high aiming accuracy in all circumstances but lack of relevant tactics). 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 multi-player online games.

Although in this paper we focus on the detection of aimbots, we believe the same approach is capable on other types of cheat that exhibit similar patterns. For example, a cheater using a map-hack in a RTSG game would go straightly toward important items before the cheater has already explored their locations; a cheater using a wall-hack in a FPS game would likely to aim around the corners frequently and shoot immediately when a target appears. As long as one can describe the behavior of a cheat in terms of the game states, then it is possible to construct a proper Bayesian Network to learn and then to detect such cheat. This type of enhancement is our on-going research work.

References

1. Admins, U.: official HLGuard discussion forum. http://forums.unitedadmins.com/index.php?showtopic=36652 (2004)
2. Admins, U.: official HLGuard discussion forum. http://forums.unitedadmins.com/index.php?showtopic=44059 (2004)
3. L. von Ahn M. Blum, N.J.H., Langford, J.: Captcha: Using hard ai problems for security. pp. 294–311
4. Balance, E.: official PunkBuster website. http://www.evenbalance.com (2007)
5. Baughman, N.E., Levine, B.N.: Cheat-proof playout for centralized and distributed online games. pp. 104–113 (2001)
6. Chen, B.D., Maheswaran, M.: A cheat controlled protocol for centralized online multiplayer games. pp. 139–143 (2004)
7. Entertainment, B.: Battle.net. http://www.battle.net (2002)
8. Eric Cronin, B.F., Jamin, S.: Cheat-proofing dead reckoned multiplayer games (2003)
9. Golle, P., Ducheneaut, N.: Keeping bots out of online games (2005)
10. Lui, J.C.S.: Constructing communication subgraphs and deriving an optimal synchronization interval for distributed virtual environment systems. pp. 778–792 (2001)

11. Margaret DeLap Bjorn Knutsson, H.L.O.S.U.S.I.L.C.T.: Is run-time verification applicable to cheat detection. pp. 134–138 (2004)
12. the New York Times: Conqueror in a War of Virtual Worlds. http://www.nytimes.com/2005/09/06/arts/design/06worl.html?ex=1283659200&en=7057c2e17780c600&ei=5090 (2005)
13. van Oortmerssen, W.: Cube. http://www.cubeengine.com (2004)
14. Pritchard, M.: How to hurt the hackers: The scoop on internet cheating and how you can combat it (2001)
15. Project, T.Z.: official HLGuard website. http://www.thezproject.org (2007)
16. Space, S.: Internet Research Reports. http://www.securityspace.com/s_survey/data (2004)
17. official website, C.: Rules 6.40 Anti-Cheat information. http://www.caleague.com/?division=csac&page=rules#6.40 (2007)
18. Wikipedia: Counter-Strike. http://en.wikipedia.org/wiki/Counter-Strike (2007)
19. Wikipedia: Valve Anti-Cheat. http://en.wikipedia.org/wiki/Valve_Anti-Cheat (2007)

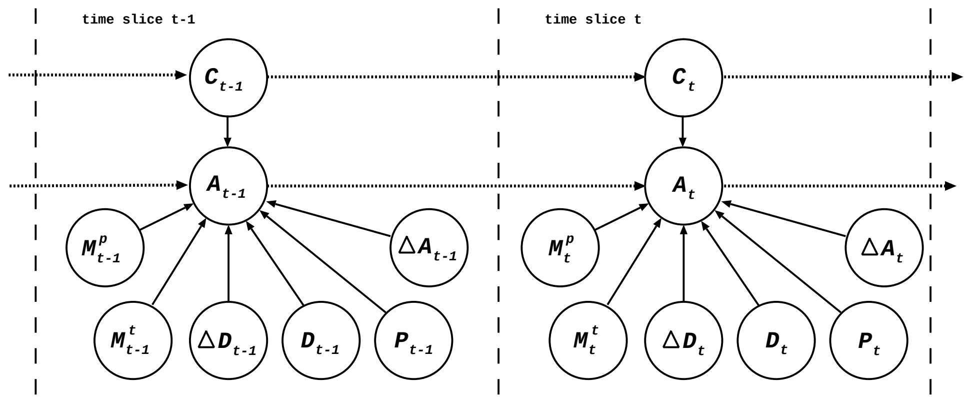
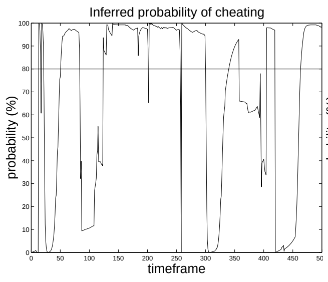
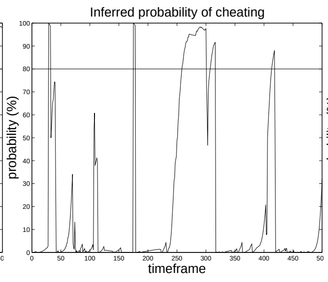


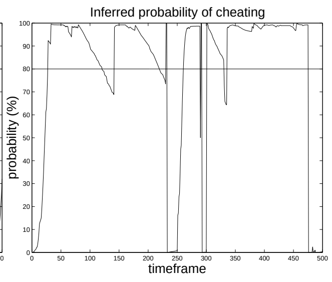
Fig. 5 The DBN used for aimbot behavior detection.



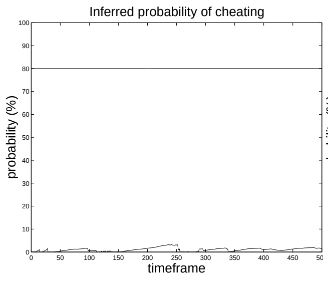
(a) cheater c_{b1}



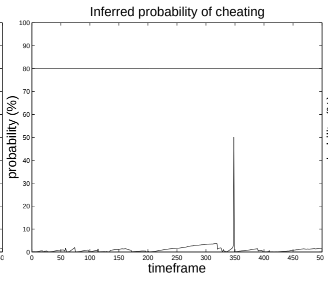
(b) cheater c_{b2}



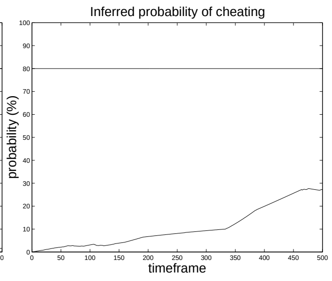
(c) cheater c_{b2}



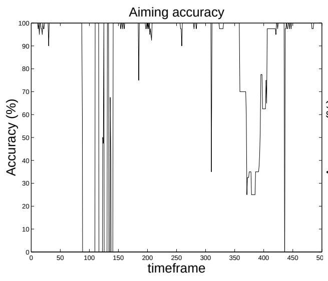
(d) honest player h_{b1}



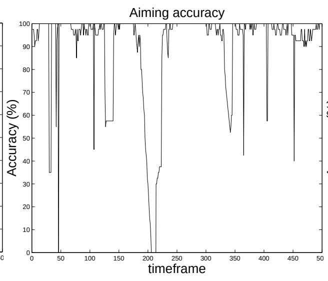
(e) honest player h_{b2}



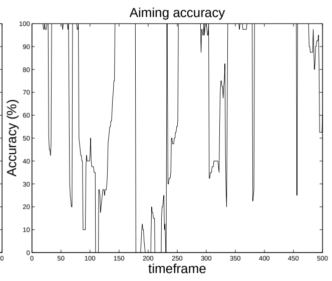
(f) honest player h_{b3}



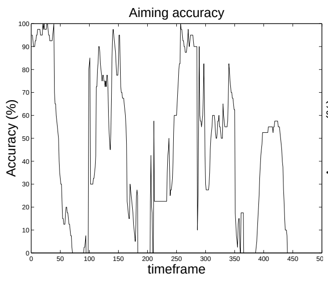
(g) cheater c_{b1}



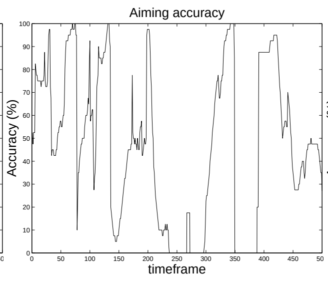
(h) cheater c_{b2}



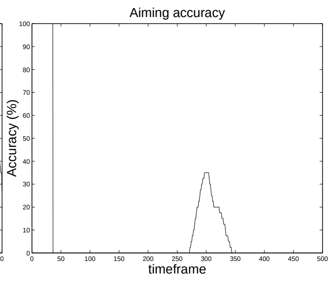
(i) cheater c_{b2}



(j) honest player h_{b1}

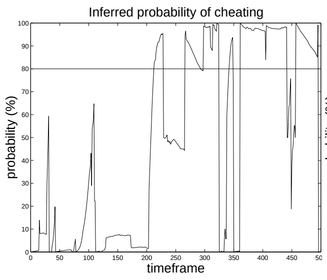


(k) honest player h_{b2}

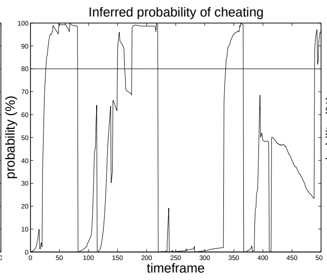


(l) honest player h_{b3}

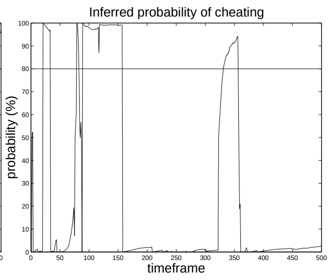
Fig. 6 Result of Experiment 1. Cheaters have probabilities fluctuating around the threshold, as illustrated in the sub-figures (a)-(c), while honest players have probabilities well below the threshold, as illustrated in sub-figures (d)-(f). For comparison, the aiming accuracies of the players are shown in the sub-figures (g)-(l) respectively.



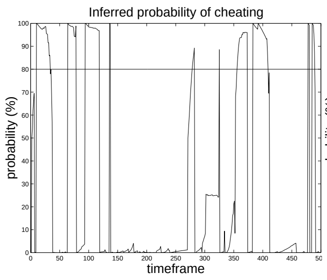
(a) auto-switching, cheater h_{c1}



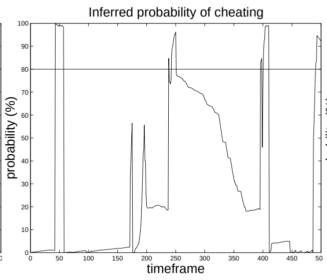
(b) auto-switching, cheater h_{c2}



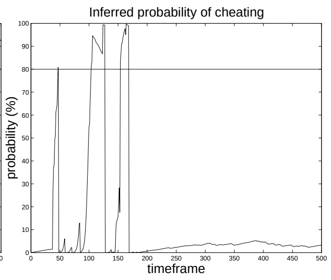
(c) auto-switching, cheater h_{c3}



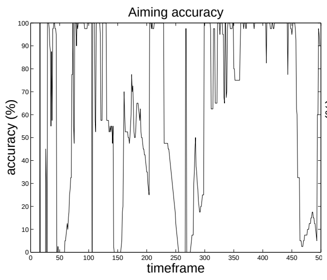
(d) intentional misses, cheater h_{c4}



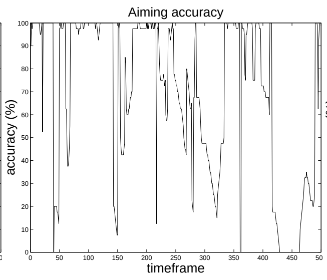
(e) intentional misses, cheater h_{c5}



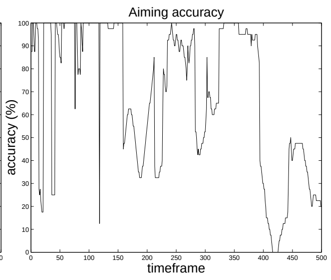
(f) intentional misses, cheater h_{c6}



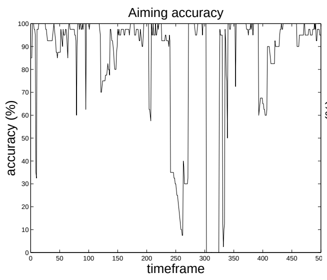
(g) auto-switching, cheater h_{c1}



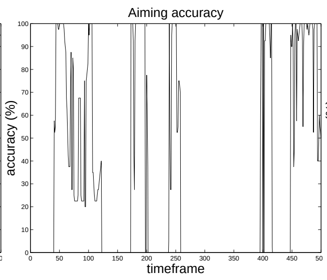
(h) auto-switching, cheater h_{c2}



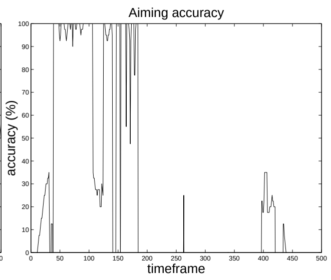
(i) auto-switching, cheater h_{c3}



(j) intentional misses, cheater h_{c4}

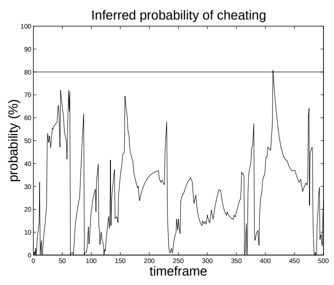


(k) intentional misses, cheater h_{c5}

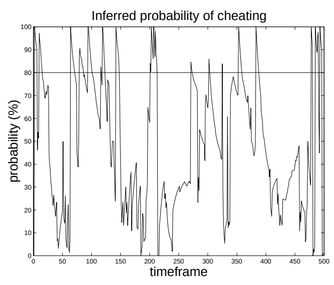


(l) intentional misses, cheater h_{c6}

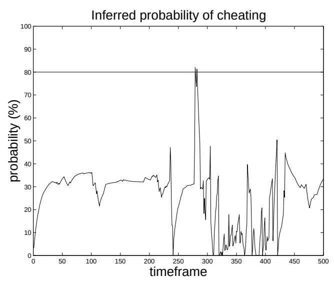
Fig. 7 Result of Experiment 2. Probability of cheaters using an aimbot that automatically switching on and off occasionally in random intervals (a)-(c), and the probability of cheaters using an aimbot that creates intentional misses in random intervals (d)-(f). For comparison, the aiming accuracies of the players are shown in the sub-figures (g)-(l) respectively.



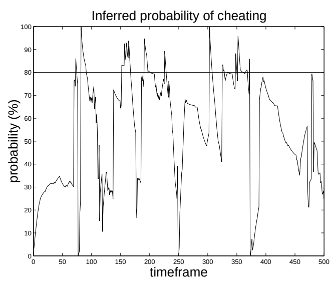
(a) honest player h_{c1}



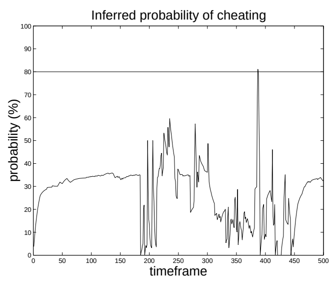
(b) cheater c_{c1}



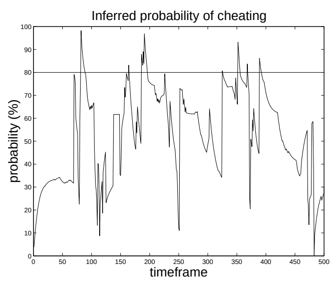
(c) honest player h_{a1}



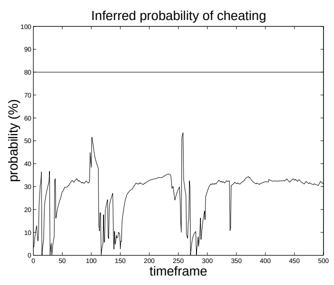
(d) cheater c_{a1}



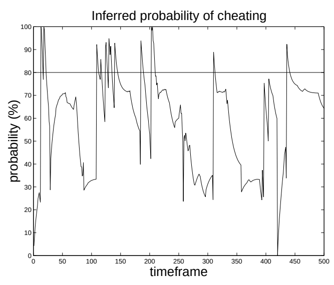
(e) honest player h_{a1}



(f) cheater c_{a1}



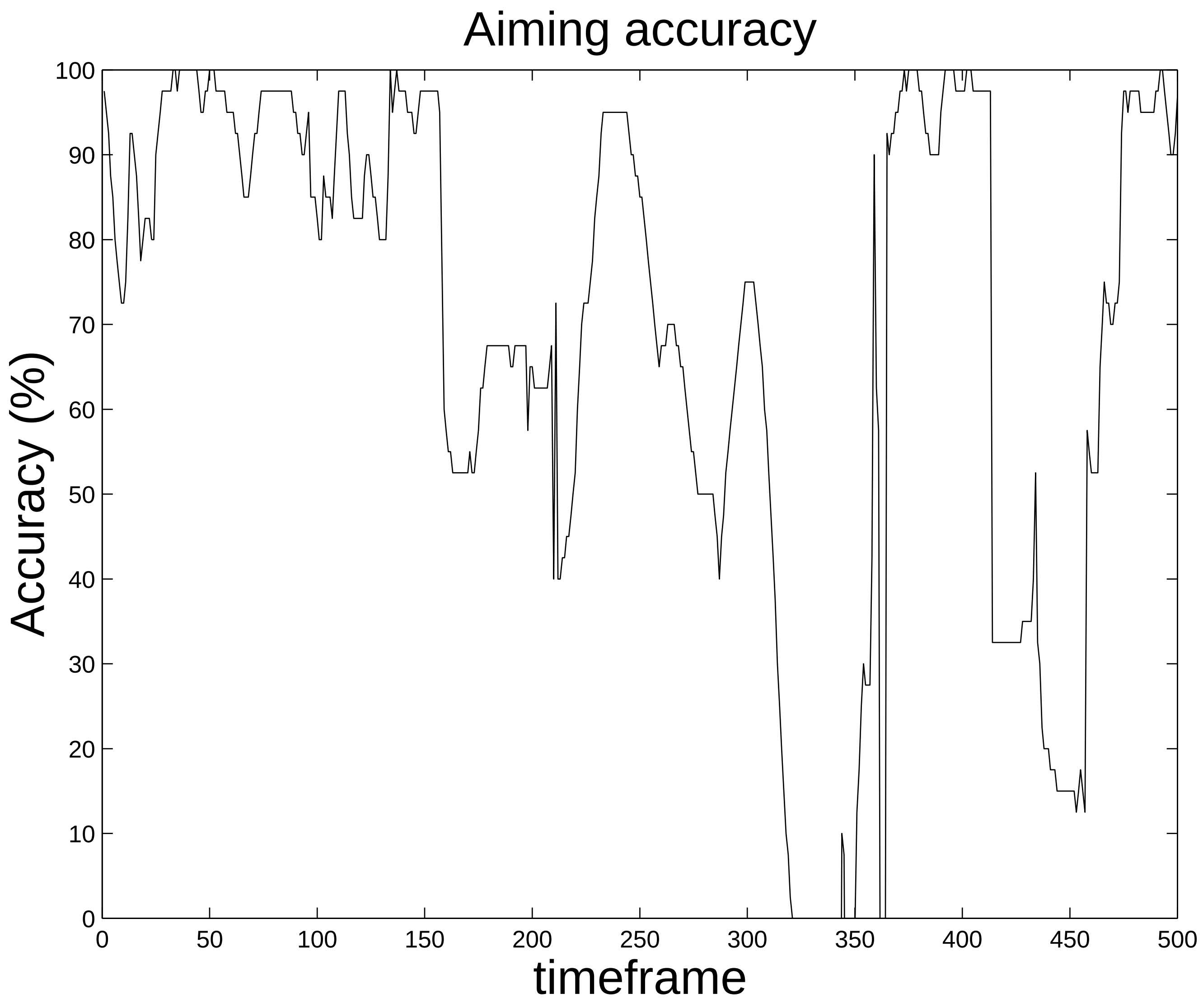
(g) honest player h_{b1}



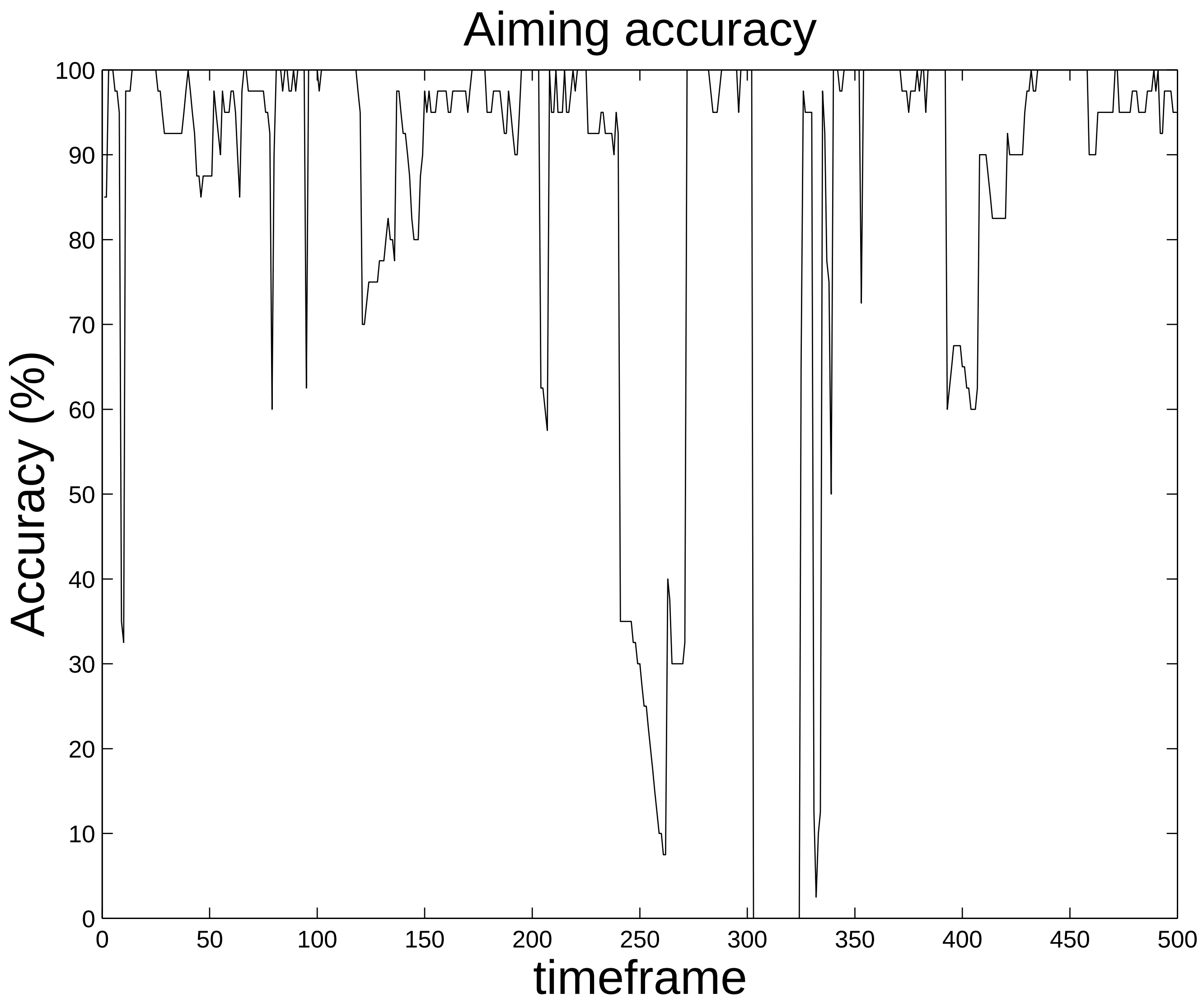
(h) cheater c_{b1}

Fig. 8 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).

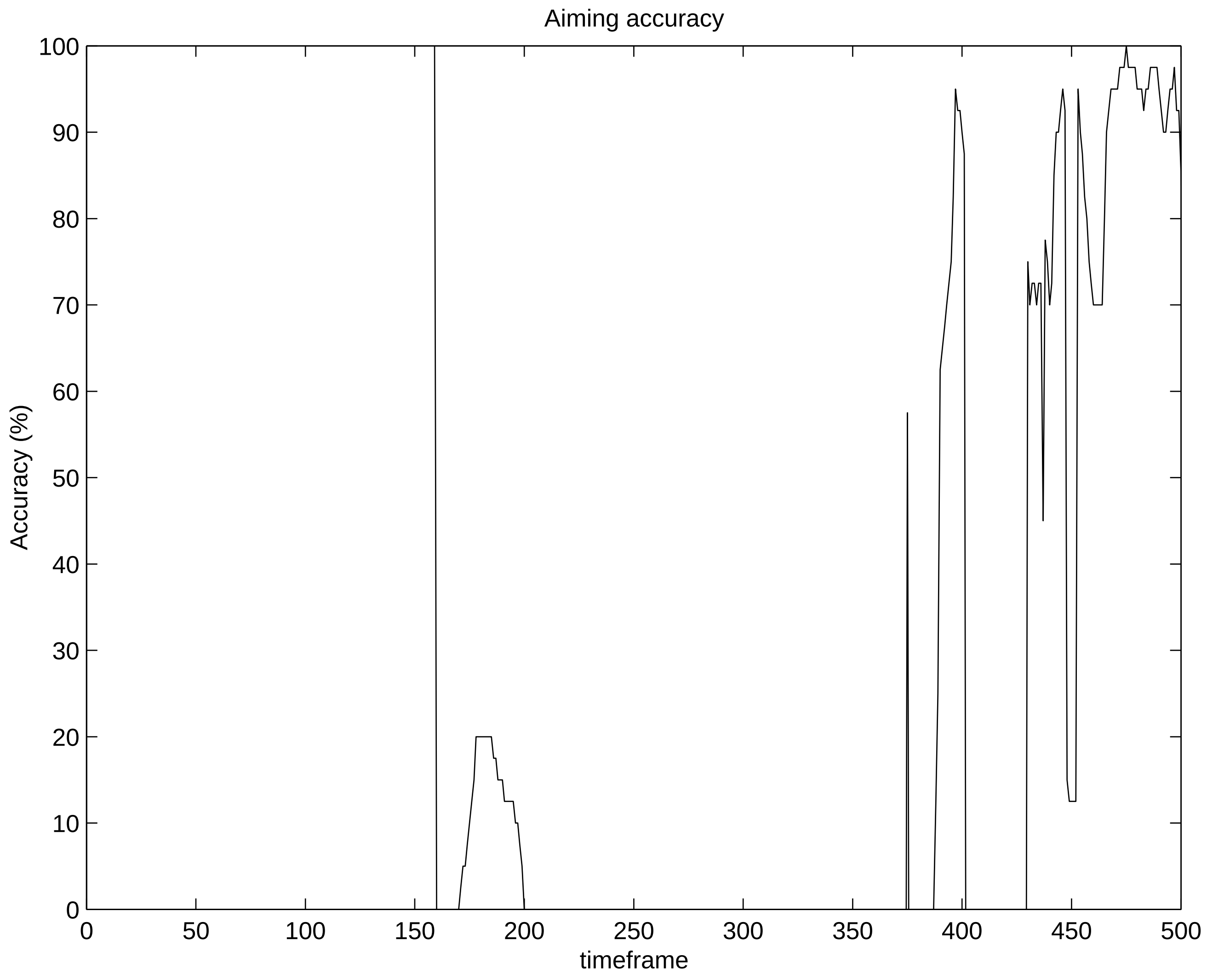
Figure


Aiming accuracy
Accuracy (%)
timeframe
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
300
350
400
450
500

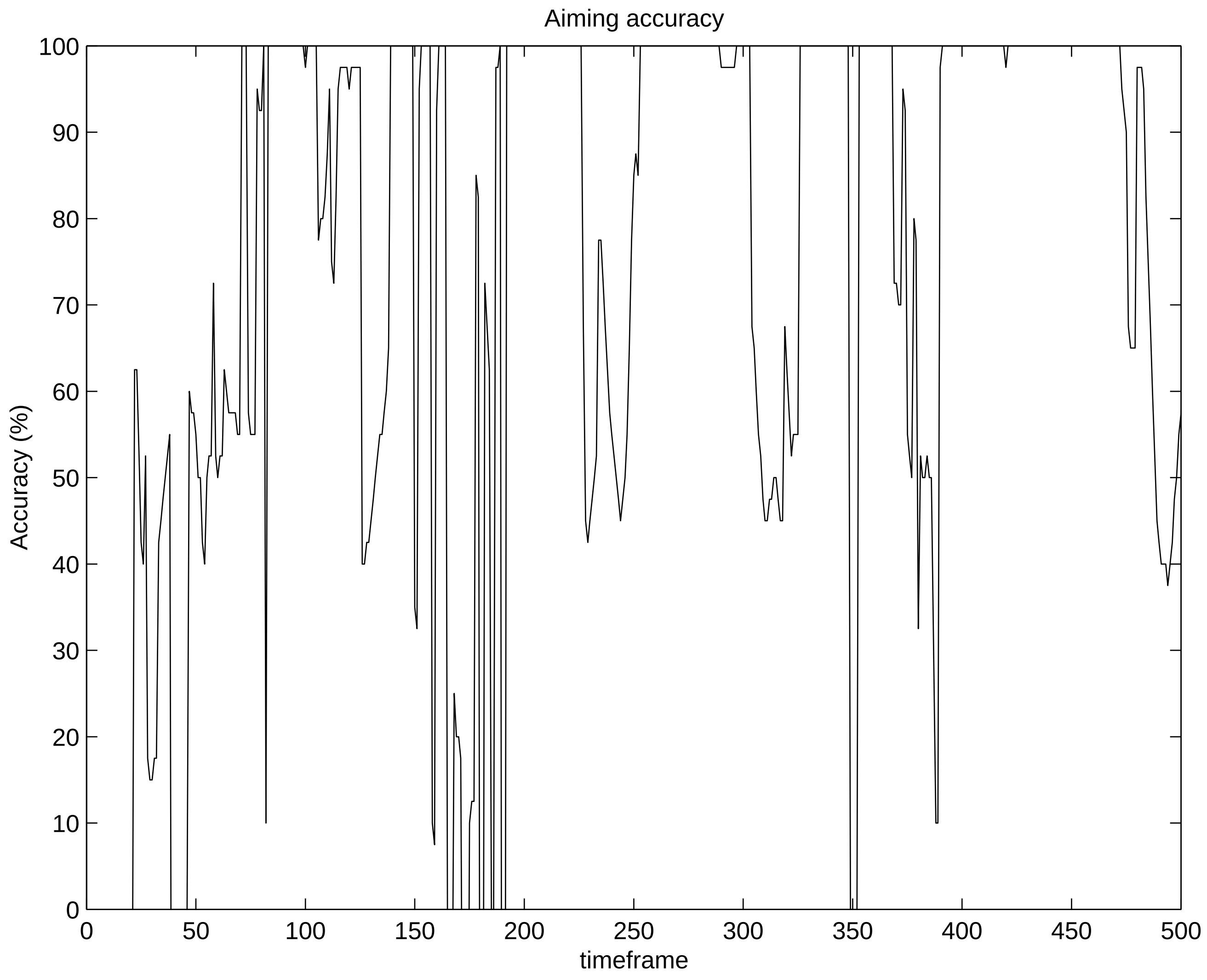
Figure


Aiming accuracy
Accuracy (%)
timeframe
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
300
350
400
450
500

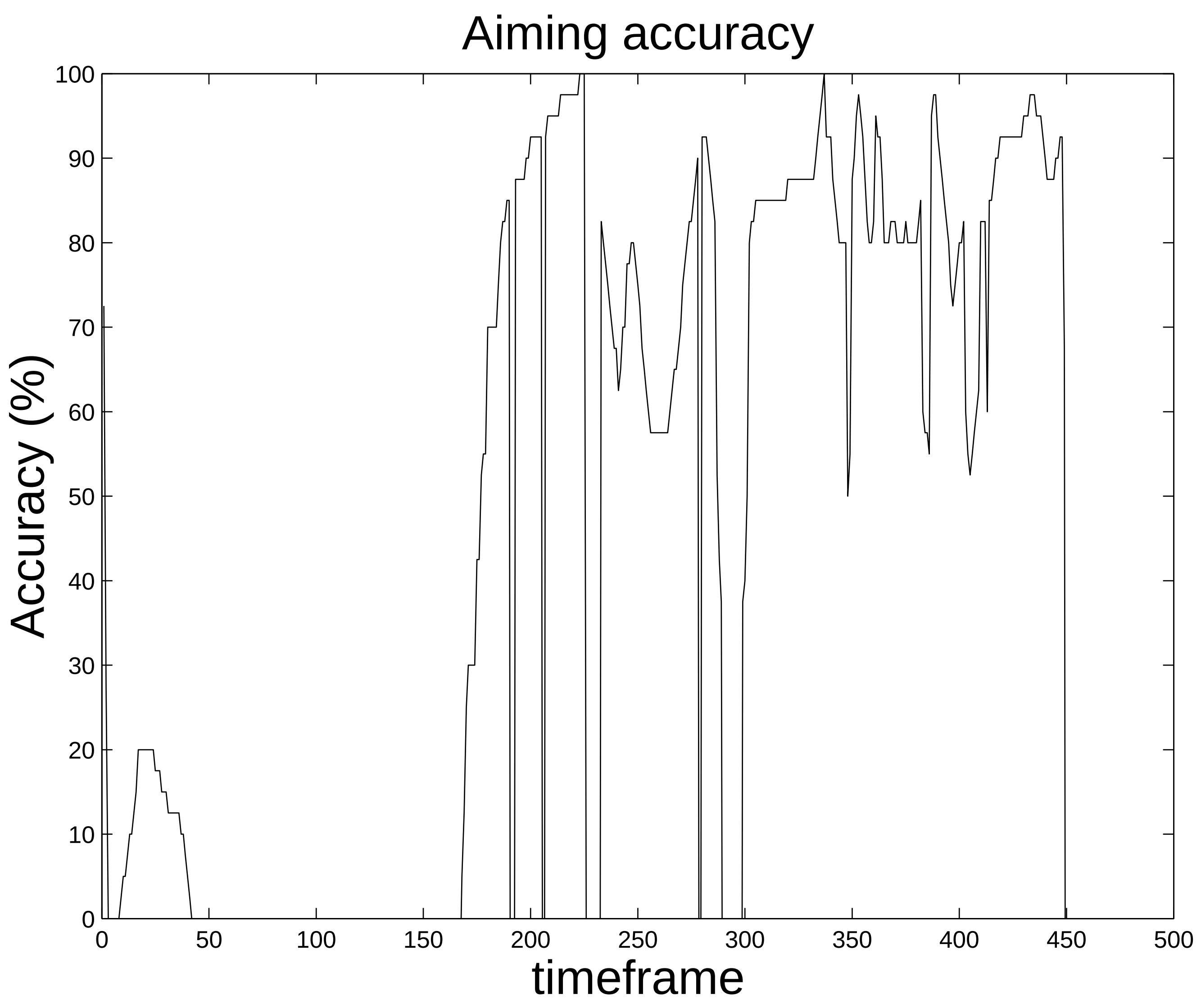
Figure


Aiming accuracy
Accuracy (%)
timeframe
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
300
350
400
450
500

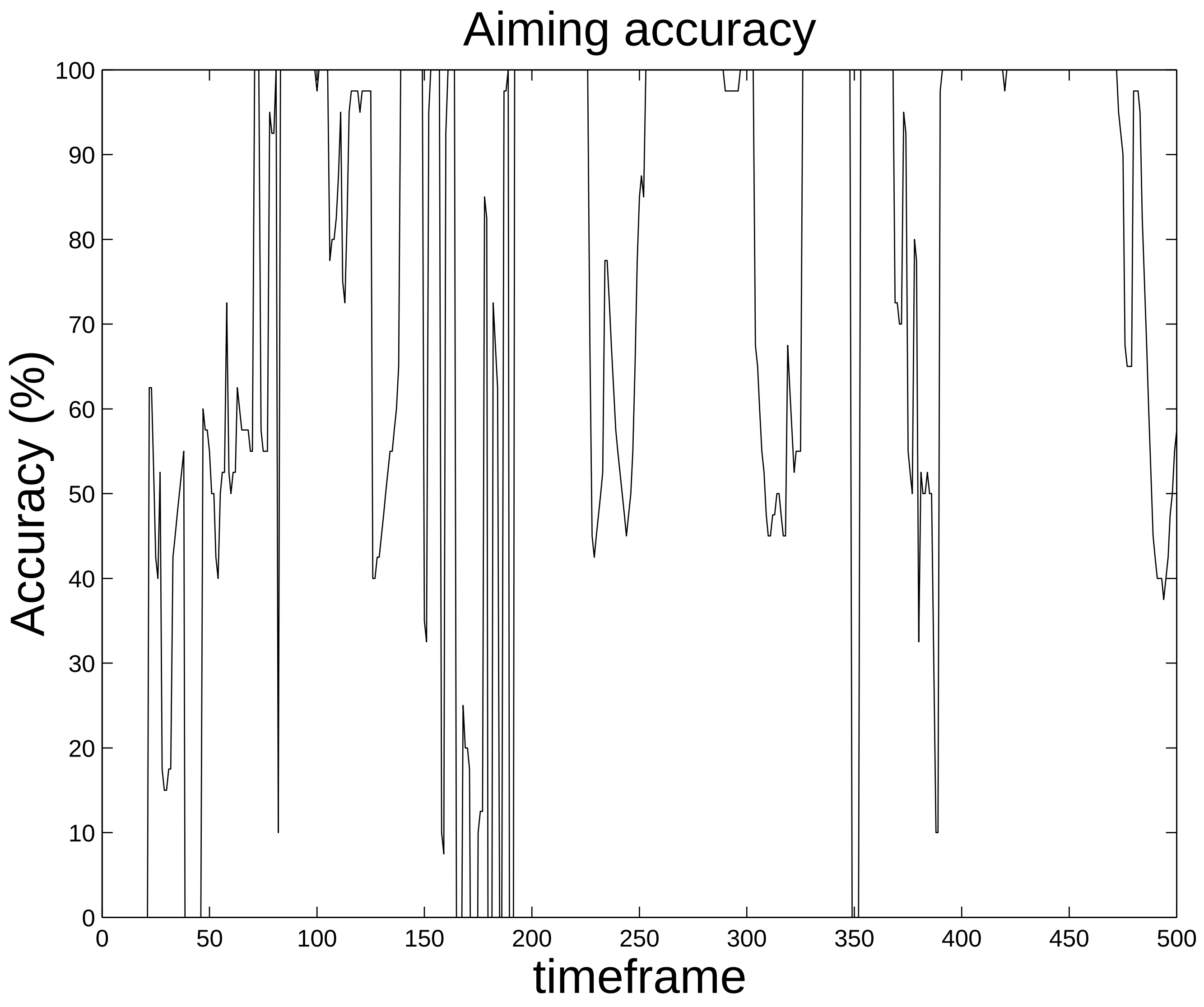
Figure


Aiming accuracy
Accuracy (%)
timeframe
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
300
350
400
450
500

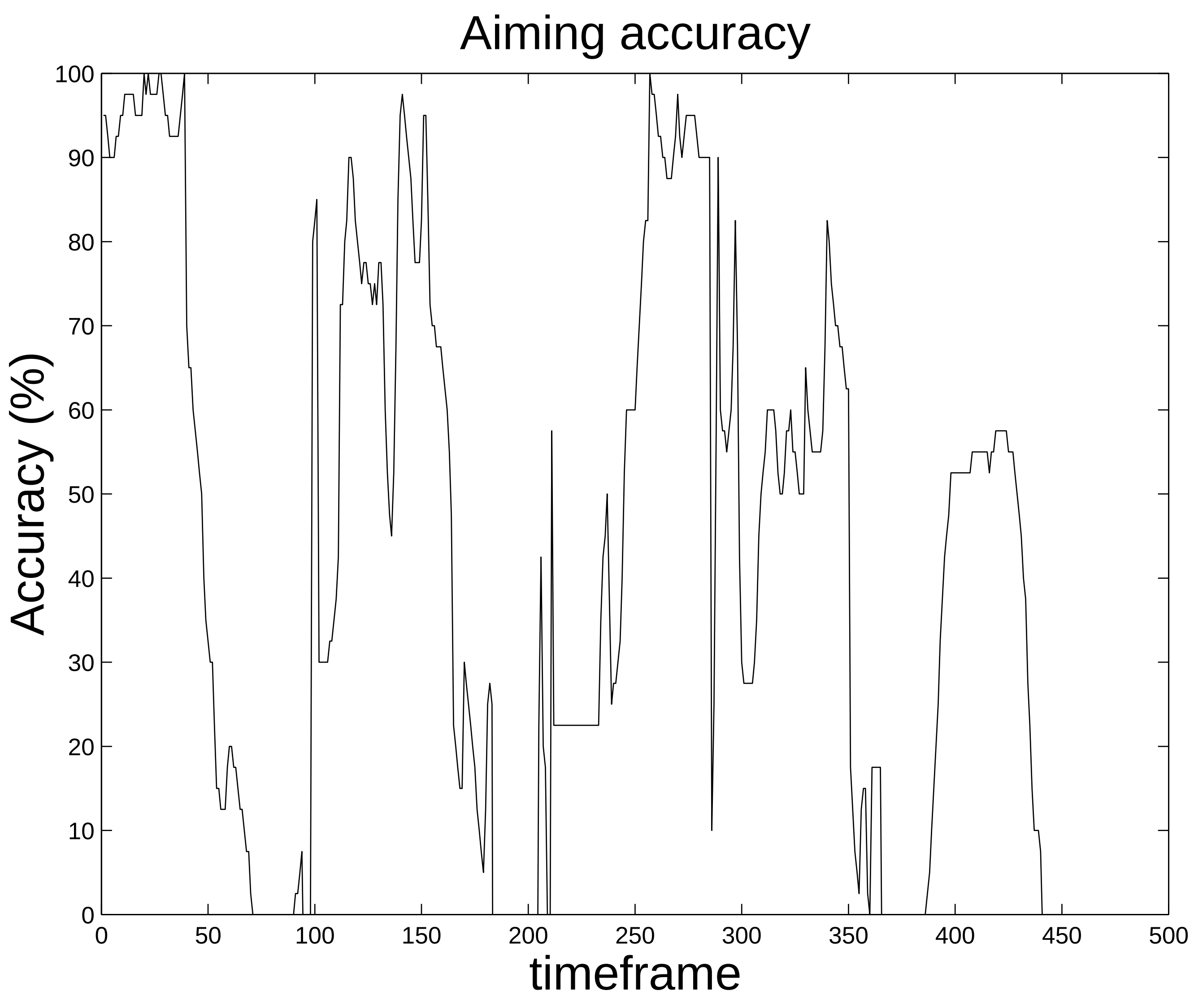
Figure


Aiming accuracy
Accuracy (%)
timeframe
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
300
350
400
450
500

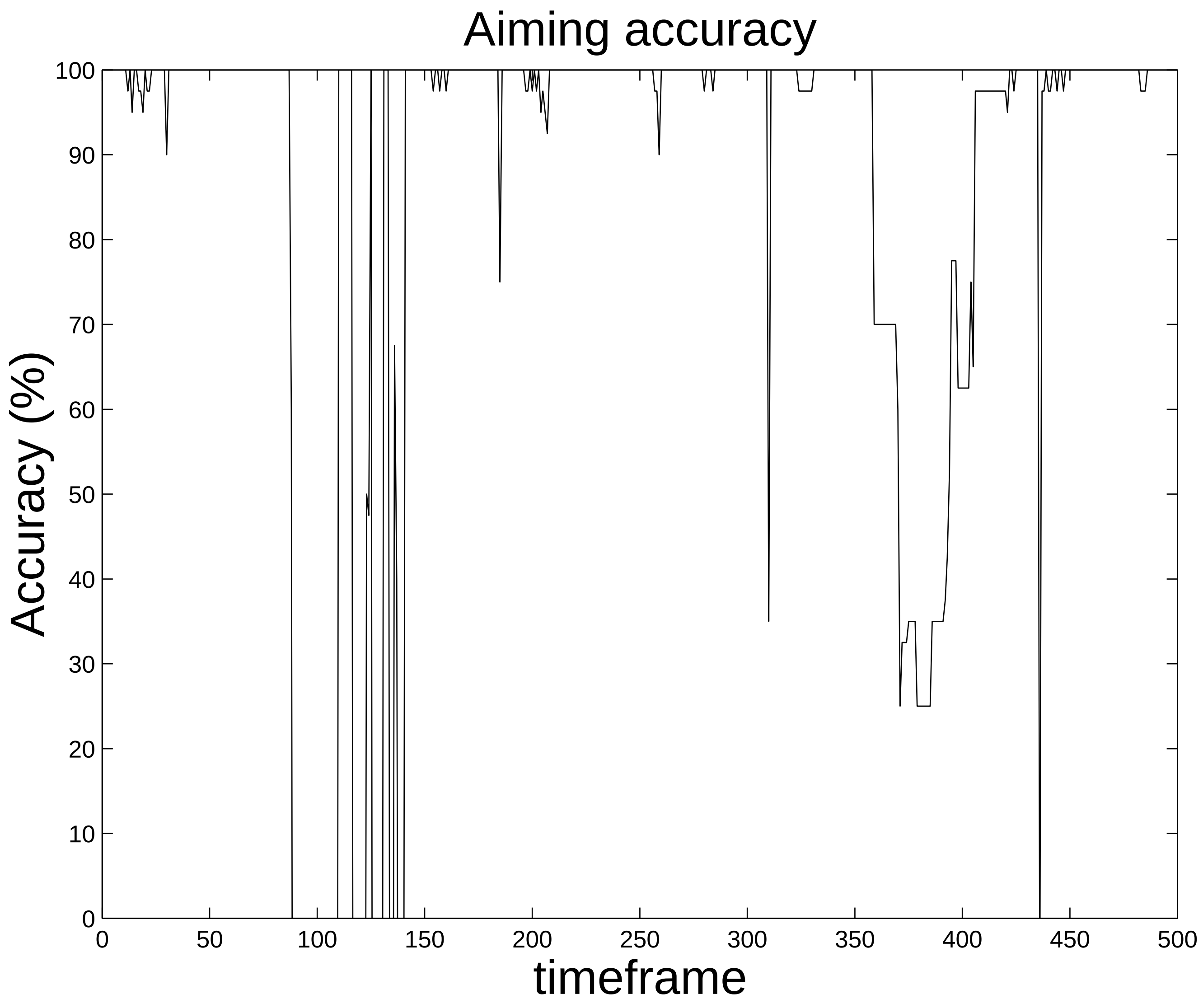
Figure


Aiming accuracy
Accuracy (%)
timeframe
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
300
350
400
450
500

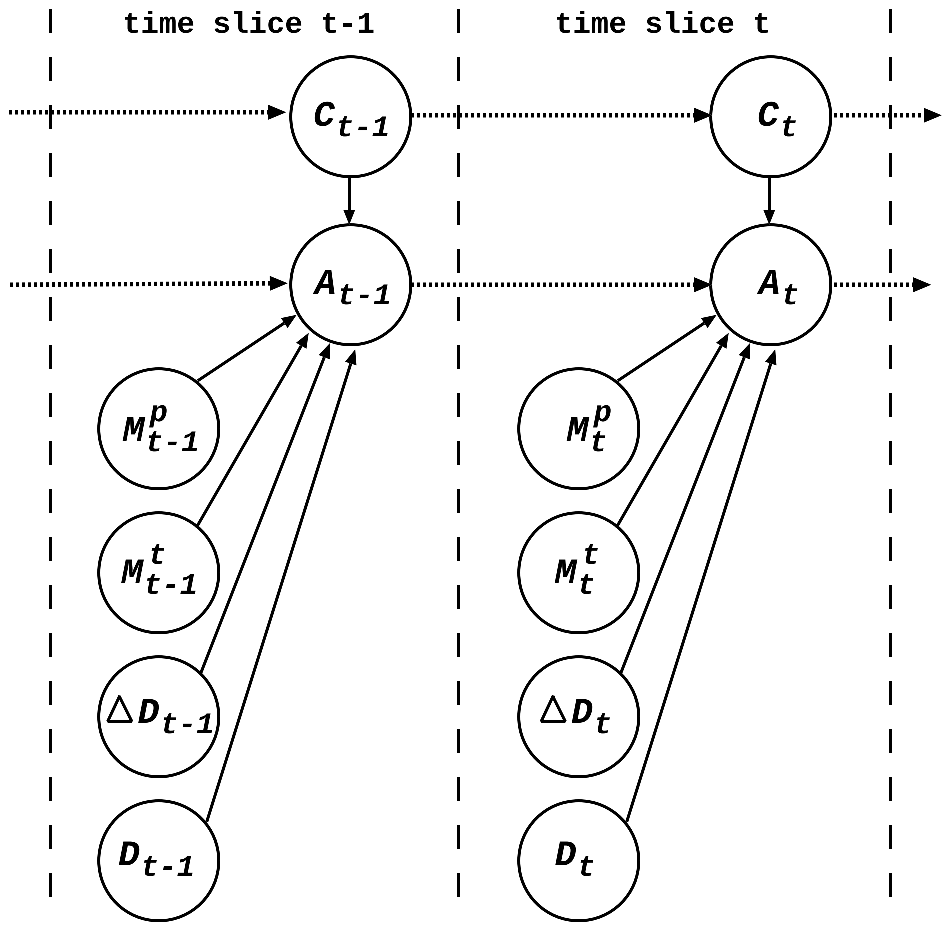
Figure


Aiming accuracy
Accuracy (%)
timeframe
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
300
350
400
450
500

Figure


Aiming accuracy
Accuracy (%)
timeframe
0
10
20
30
40
50
60
70
80
90
100
0
50
100
150
200
250
300
350
400
450
500

Figure


time slice t-1
time slice t
C_{t-1}
C_t
A_{t-1}
A_t
M^p_{t-1}
M^t_{t-1}
ΔD_{t-1}
D_{t-1}
M^p_t
M^t_t
ΔD_t
D_t

* authors' response to reviewers' comments
Click here to download authors' response to reviewers' comments: reply1.pdf

Reply to reviewer 1

Paper Title: Dynamic Bayesian Approach for Detecting Cheats in Multi-Player Online Games

Authors: S.F. Yeung and John C.S. Lui

Thank you for your comments on our paper. Your comments helped us to make our paper more precise, and also clarify some points which may be confusing to readers. We have made a *moderate revision* on the paper. In this reply, we have two sections. Section 1 is a summary of our revisions on the paper. Section 2 is a reply to the individual questions raised by the reviewer.

Note: *all reference number below are based on the original submission, so as to be consistent with reviewer comments.*

1 Summary of revisions

1. Some typo and grammatical errors are corrected.
2. Minor modification on the Abstract to state clearly that our experiment results are based on a first-person shooter game.
3. A new pagragraph in Section 4.2 to discuss the motivation of our detection scheme.
4. The sub-section Computational Analysis in Section 4.4 is expanded to include more details.
5. Details of the players involved in the experiments are included in the Results section.
6. A new section "Future Work" is added to discuss the ideas of possible enhancements of our approach.

2 Reply to reviewer's comment

1. **Reviewer:** Missed this on the first go around. Section 2 starts with a mistake (MMORPS)

Reply: Thanks for pointing out the mistake, we have also corrected some other typos and grammatical errors.

*** authors' response to reviewers' comments**
Click here to download authors' response to reviewers' comments: reply2.pdf

Reply to reviewer 2

Paper Title: Dynamic Bayesian Approach for Detecting Cheats in Multi-Player Online Games

Authors: S.F. Yeung and John C.S. Lui

Thank you for your comments on our paper. Your comments helped us to make our paper more precise, and also clarify some points which may be confusing to readers. We have made a *moderate revision* on the paper. In this reply, we have two sections. Section 1 is a summary of our revisions on the paper. Section 2 is a reply to the individual questions raised by the reviewer.

Note: *all reference number below are based on the original submission, so as to be consistent with reviewer comments.*

1 Summary of revisions

1. Some typo and grammatical errors are corrected.
2. Minor modification on the Abstract to state clearly that our experiment results are based on a first-person shooter game.
3. A new pagragraph in Section 4.2 to discuss the motivation of our detection scheme.
4. The sub-section Computational Analysis in Section 4.4 is expanded to include more details.
5. Details of the players involved in the experiments are included in the Results section.
6. A new section "Future Work" is added to discuss the ideas of possible enhancements of our approach.

2 Reply to reviewer's comment

1. **Reviewer:** I would like to see a discussion of the increased computation load on the server for the scheme, specifically as it relates to determining the player's aim point. Under normal operation a FPS server only needs to do a raycast when a player fires an instant-hit weapon, but for

this detection, the FPS server must do a raycast for every player every frame in order to determine a player's aiming target. This seems to be the dominant computational cost in the scheme.

Reply: Thanks for your suggestion. We include the following discussion in our revision, in the sub-section Computational Analysis in Section 4.4.

Let us have a brief discussion on the computational workload of using the proposed scheme on the server. In commercial games such as counter-strike and Quake, the game clients send all commands (keyword inputs) to the server. The server runs the simulation of the whole game because all game states must be authorized by the server before the server broadcasts the information to other players. Therefore, the server already knows the position and the aiming direction of all players at each instance of the game. In our proposed method, the only additional computation requires to perform is to compute the "distance between a player's aiming trajectory and the player's current target", we called it the player's accuracy. This is to compute the distance between a point and a line which involves a cross-product operation. Moreover, as we mentioned before, the computation only occurs when there is any visible target, and the server only needs to trace the accuracy with respect to the current target, until it is not visible to the player or their distance is beyond a certain threshold. Also, for FPS games the number of players involved in the same game session typically varies from 8 to 32. Hence, the computational overhead induced should not be much. Moreover, in practical implementation one could create an individual queue to store a copy of the user states, and then carries out the computation of the player's accuracy and also the Bayesian inference routine in a computational thread, which is separated from the main server loop so as to reduce the real-time workload on the server.

* authors' response to reviewers' comments
Click here to download authors' response to reviewers' comments: reply3.pdf

Reply to reviewer 3

Paper Title: Dynamic Bayesian Approach for Detecting Cheats in Multi-Player Online Games

Authors: S.F. Yeung and John C.S. Lui

Thank you for your comments on our paper. Your comments helped us to make our paper more precise, and also clarify some points which may be confusing to readers. We have made a *moderate revision* on the paper. In this reply, we have two sections. Section 1 is a summary of our revisions on the paper. Section 2 is a reply to the individual questions raised by the reviewer.

Note: *all reference number below are based on the original submission, so as to be consistent with reviewer comments.*

1 Summary of revisions

1. Some typo and grammatical errors are corrected.
2. Minor modification on the Abstract to state clearly that our experiment results are based on a first-person shooter game.
3. A new pagragraph in Section 4.2 to discuss the motivation of our detection scheme.
4. The sub-section Computational Analysis in Section 4.4 is expanded to include more details.
5. Details of the players involved in the experiments are included in the Results section.
6. A new section "Future Work" is added to discuss the ideas of possible enhancements of our approach.

2 Reply to reviewer's comment

1. **Reviewer:** There are a number of simplifying assumptions made in order to make the solution tractable. Notably, the dependency on determination of cheating seems to be very specific to a

FPS game, the exact weapon used in the game and a specific player. For example, accuracy of a weapon would largely depend upon the precision of the weapon (i.e. a shotgun and machine gun being less precise than a sniper rifle). In addition, the skill of the player will matter, too, in terms of accuracy with some being better than others. And the map, where some battles are all done at a distance and others are all close up. In short, trying to apply a single, acceptable threshold for cheating to a heterogeneous (in terms of skill and weapon and map) bunch of users seems like it would fail in many cases. They also provided a weak response to my comment:

The paragraph right before section 4.3 is very speculative. Claiming that an aimbot will "always" exhibit some kind of pattern seems hopeful, not based on reason. Saying that a player that is good must be good at both offense and defense also seems speculative. In my personal experience, some players have an offensive style of play and may be quite bad at defense, and vice versa. To make such claims, the authors would need to profile the experience and style of play of many players.

Basically, they tried to justify including the prose they have. While their reasoning may, indeed, be sound, their claims read like such results are "proven" or "facts". This is not good science. It would be much better if their prose was qualified or written as assumptions.

Reply: Thanks for your comment. We have re-written our paper, including the paragraph right before section 4.3, based on your comment. In particular, we improve the presentation by adding the following nodes into our Bayesian network, in the Future Work section:

1. weapon - the player's current weapon
2. map type - discrete values represents close up battle, long range battle or in-between
3. no. visible targets - the number of targets that currently visible to the player
4. action - the player's current actions, such as dodging, jumping, standing, walking or running
5. skill - a value indicate how good the player should be, discuss below

Since the inspiration of our approach is to imitate how professional administrators detect cheaters by observation. The motive of adding these new nodes is trying to capture the skill of the player and capture more information of the whole game. The proposed "skill" node could be obtained by a centralized method. For example, many popular FPS games have regular national or international tournaments such as the Cyberathlete Amateur League (CAL) tournaments. From the concern you mentioned, we may think that one of the most critical issue is to differentiate CAL-level players from cheaters. Since nowadays most popular FPS games use global login ID to identify players, which can only be obtained by purchase a new copy of the game. And there exists centralized global databases for all servers to retrieve and save banned login ID. Therefore, it is possible to query a player's tournament history according to the global login ID and determinate the player's "skill" accordingly.

Also, we have added the following paragraphs at the end of Section 4.2.

To avoid being detected due to its high aiming accuracy, some advanced aimbots may pretend to act as a normal player, either by automatically switches itself on and off periodically, or by creating some intended misses from time to time. However, human players have diverse behaviors but cheaters using aimbots are very likely to exhibit some kind of patterns in their cheating behaviors. For example, a skillful player who is good at shooting will adopt a specific tactic to gain the highest advantage according to the current circumstance and landform of the virtual world. However, a cheater may exhibit outstanding accuracy but lack the sense of paying attention to the environment of the virtual world. An experienced administrator can tell whether a player is skillful or cheating by carefully reviews the recorded game. In most cases the cheaters would configure the aimbots such that they would perform much better than the cheaters themselves when they are unassisted. However, if a cheater configures the aimbot to make only a slight improvement in performance, e.g., by automatically turning on occasionally or even rarely, there are still clues that one can observe the presences of an aimbot at the very moment when the aimbot is activating. For example, the most advanced aimbot today is called the "charged-type" aimbot, these aimbots are turned off by default but will only activate when the cheater presses a self-defined hotkey, and then turned off immediately once the cheater releases the hotkey. More advanced aimbots would only activate when the crosshair is sufficiently close to the target, makes a more natural look therefore harder to detect. These aimbots are completely unnoticeable in peacetime but may still be caught when they are activated, although very careful and professional reviews are required. These are the reasons why the most authoritative cheat detection method used today is still by human observation, and it is especially important in major tournaments [1] and [17]. Note that above are observations we made in the FPS type of game. Based on these observations, we are able to make the necessary Bayesian model to detect aimbot. We believe the proposed approach holds promise in other forms of games or cheats, although one may need to make some specific observations and derive new Bayesian models for different types of games.

2. **Reviewer:** While they did address my comment:

   For the results, who where the players? The authors? That seems a biased group. If not, then how were they solicited? What was there skill and style of play? These comments tie into my above concerns about the general applicability of this to a wide range of game/player settings.

   They only did so in their response. Their prose regarding these details in the paper seems to be unaltered. Again, this lack of detail is bad science. And, overall, I don't want the authors to convince me in their responses. Rather, they should add appropriate text to their paper to convince the readers of their article.

   Reply: Thanks for your comment. We have included the content in the previous reply, i.e. the details of the players, in this submission as you suggested. We also have a minor modification on the Abstract to state clearly that our experiment results are based on a first-person shooter game.

   Moreover, as you concerned, and also due to the new nodes proposed above, there is a need for a new and larger scale experiment. We have once tried to organize a large scale experiment in Newcastle University, but we hope that we could deploy the Bayesian framework in a public commercial gaming environment. However, supports from server companies are required and we think that we may not be able to include the new results in this submission.

3. **Reviewer:** The English needs to be cleaned up before this appears in print. The new text added is especially problematic in places.

   Reply: Thanks for your comment. We have proof-read the paper again for this submission.