How to deal with bot scum in MMORPGs?

Sylvain Hilaire
School of Computer Science
and Engineering
Seoul National University
Seoul 151-744, Korea
Email: sylvain.hilaire@mines-saint-etienne.org

Hyun-chul Kim
School of Computer Science
and Engineering
Seoul National University
Seoul 151-744, Korea
Email: hkim@mmlab.snu.ac.kr

Chong-kwon Kim
School of Computer Science
and Engineering
Seoul National University
Seoul 151-744, Korea
Email: ckim@snu.ac.kr
Tel/Fax: +82(0) 2-880-6582/2-871-4912

***Abstract*—Nowadays, bots are becoming a critical issue for the online gaming world. Bots give unfair advantages, and are then considered as cheating and undesirable on game servers. Currently, CAPTCHA and human controls are the most commonly chosen strategies to catch bots. However, these methods are intrusive and complicated have proven to be inefficient due to the large servers' populations. Researchers have proposed various kinds of automated detection scheme. Yet, these proposals exhibit unpractical features, such as complexity or scalability issues, making the deployment on real systems problematic. We propose to study bots' and humans' client-server communication patterns, focusing on a one of most famous MMORPG called *World of Warcraft*. Intuiting that, for sake of efficiency and human-looking behavior, bots cannot constrain both packet timing and sizes, we propose a detection scheme that combines both parameters. We propose an online algorithm that processes our scheme on the fly as packets arrive. We evaluate the proposed scheme with real packet trace and observe that it can detect bots with small false alarm probability.**

I. Introduction

Nowadays, bots are becoming a critical issue for online gaming world. Used to perform automatically boring tasks and gain unfair benefits, they are cheating and undesirable on servers. Moreover, their number remains unknown and probably keeps growing as the MMOGs (massively multiplayer online games) become more popular, resulting in huge game imbalances, great bothersome for honest players. They discourage players from keeping paying subscription fees and drive out bored people, which naturally threatens game companies' profits. To fight this scourge, companies' engineers must react quickly to bots' evolution so as to limit their spread while avoiding wrongly accusing human players of cheating. They mostly use repeated Turing tests - including CAPTCHAs and human controls -, which exhibit very accurate decision and reasonable deployment requirements. However, their position is mostly reactive and the best they can do is flow containment. On the other hand, academic researchers are interested in deeply understanding bots' and humans' very specific behavioral features. This scientific approach yields enduring and automated detection methods. Yet, each work focuses on a couple of specific rationales rather than considering a global and real-time defense strategy. Besides, proposed schemes exhibit unpractical scalability and complexity characteristics, making their deployment on real systems problematic.

We provide first a deep survey of the bots detection field to clear a ground for our work. Then, studying traffic patterns, our main idea is that bots' information exchange patterns are somehow constrained resulting in a duality of information timing and amount, and a pitfall for bots. From this, we propose a real-time detection method for ground layer character preselection and test it with *World of Warcraft* (a.k.a. *WoW*) traces. So, our problem consists in detecting bots using short traces samples and performing simple calculations only, for sake of speed. Finally, we propose a big plan to integrate the proposed scheme in a global defense system that could be a future work.

The remainder of this work is organized as follows. Section 2 reviews related works and draws a general picture of the current bots detection's state of the start. Section 3 discusses trace collection. Section 4 explains our strong ideas and describes features derived from these ideas and our observation of traces. Section 5 describes the tests using the proposed features and corresponding algorithms, to come up with an online detection scheme. Section 6 presents and discusses our results. Section 7 explores some important issues and possible future directions. And section 8 contains our conclusion.

II. Related work

Numerous strategies have been proposed for anti-bots defense. Installing software on client machines to prevent bots usage is common but helpless against well designed bots. Turing tests ([15]) include CAPTCHAs and human controls. CAPTCHAs ([2], [3], [4], [5], [6], [7]) yield almost perfect decision. But Turing tests might be intrusive, exhibit high complexity and cannot deal with large populations. Traffic analysis approach ([1]) offers low time complexity and good results, but is not yet on online fashion. Despite its power and robustness, I/O devices event sequence analysis ([8]) exhibits many unpractical characteristics. Finally, there are a plethora of schemes based on payload check: trajectory ([11], [10], [9]), ON-OFF activity ([12]), action type's sequences ([13]) and aiming accuracy analysis. It exhibits very good performance but also high complexity, limited portability and sometimes raises privacy concerns.

Table I summarizes and compares bot detection schemes. The chosen criteria are standard performance metrics (accuracy, false alarm rate and detection time) and features concerning each scheme's practical applicability (e.g. robustness,

TABLE I: Survey of the game bots detection research area

Scheme/Technique	*Turing test*	*Anti-cheating software*	*Traffic based*	*I/O devices event*	*Payload based detection schemes*			
					Trajectory	*Activity patt.*	*Farming (RGP)*	*Aim bots (FPS)*
References	[2], [3], [4], [5], [6], [7]	-	[1]	[8]	[11], [10], [9]	[12]	[13]	-
Accuracy	above 99%	-	90% to 95%	95% to 99%	95% to 99%	90% to 95%	90% to 95%	-
False alarm rate	<1%	-	<1%	-	RPG: 0%	-	-	-
Robustness against human activity mimic	not behavior based	not behavior based	medium to strong	strong	very strong	medium	strong	quite strong
Scheme deployment	server	client	client/server	client	server	server	server	server
Required traces/logs	none	none	traffic traces	window events sequence	movement traces	movement traces	ID, action and time logs	movement, aiming direction and distance
General applicability	universal	game / game type	universal (limited by feature set)	large considering games but prefers certain bot types	functionality	universal (limited by feature set)	game type	game type
Detection time	nul	nul	9 to 45min	-	FPS: 3min20 to 12 min; RPG: 12 to 60 min	20 min	30 to 45min	-
Specific limitations	possible harm to game experience, complexity	countered by standalone bots or smart programming	-	scheme stuck at client side	difficulty to understand deeply features properties, prediction power and interaction; high complexity; limited portability (fit to game); privacy concern			

generality or possibility to deploy at server side). Besides, we claim that robustness against evasion is achieved, when detection evasion harms bot's performance so much that it is worthless. In addition, we observe that false alarm rate is rarely taken into account, though it is very important in practice. Consequently, we will keep an eye on it in our study.

Eventually, it seems hard to achieve high accuracy, reasonable detection time and realistic time complexity at the same time. Moreover, since many schemes have been already offered, a combination of them could achieve better overall performance. We believe that human check always yields the best final decision while automated Turing tests offer a precious assistance to select suspects for further analysis. However, because Turing tests also need help to overcome game servers' crowd, we propose an online traffic pattern based detection method as ground layer classifier, for sake of relatively low complexity and hence good scalability.

III. Data collection

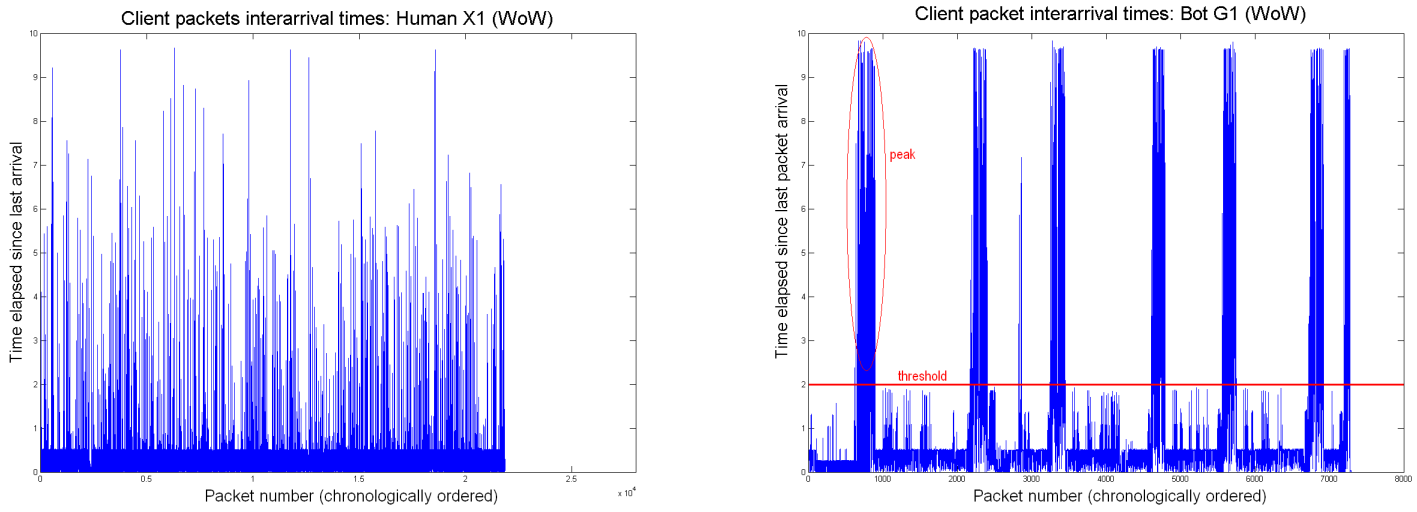
In order to design our online detection scheme, we collected *WoW* traces at client side for both human players and bots. We chose two bot programs representing fully autonomous and semi-autonomous bots, respectively *Mimic* (traces M1 and M2) and *Gatherbuddy* (traces G1 and G2). Their popularity and the possibility to get information from the related communities' forums were other choice criteria. On the other hand, we gathered four human players, including beginners (traces SS1 and S1) and experts (traces SC1 and X1). All players (bots and humans) were performing hunting with the same character for sake of fair comparison. Bots were facing each favorable (traces x2) and unfavorable conditions (traces x1) related to relief and monsters' dangerousness, so that we got trace diversity and a possibility to check the robustness of our proposed scheme. Using the *Wireshark* tool, we collected then eight traces of about two hours each. Finally, we extracted from them plain text files containing for each packet its arrival time, its TSDU length.

IV. Observations and ideas

In this section we gather our thoughts regarding the detection method and features related rationales. We also present an insight of our traces and some observations that will serve as a basis for the detection criteria definition.

A. Underlying ideas

While many good bot detection schemes have been proposed, almost nothing is dealing with anti-bots countermeasures. However, he who manages to detect a cheater must also enforce some really dissuasive punishment policy. Nowadays, temporary and definitive ban are the available solutions but are somehow irreversible, making multiple checks and parsimony necessary to avoid catastrophic wrong sentences. Thus, countermeasures are still rarely used, encouraging bot users to have a try with limited risks. We rather believe that reaction should be immediate, increasing in intensity while suspicion is getting stronger. One basic component of such a defense mechanism is a ground layer detection method that investigates exhaustively and on-the-fly every player's traces to yield a pre-selection of



(a) Human player: trace X1

(b) Bot player: trace G1

Fig. 1: Client packets interarrival times

"good candidates". Updating metrics at each packet arrival, the online detection tool finally makes a decision when it has collected enough data.

On the other hand, online detection raises time and space complexity's issue. Indeed, it must operate between two packets' arrivals while devoting limited storage capacity to each player for sake of scalability. For this reason and its generality, we chose **client traffic analysis based detection**. Though this class of methods has exhibited in [1] some breach for anti-detection, if a bots developer has to sacrifice efficiency to achieve stealth, we will claim victory. Besides, we noticed that bots tend to send less information than humans for similar tasks, theoretically resulting in lower data rates. Actually, we also observed for *Mimic*'s case that generating random information is dangerous for bots: it makes them behave stupidly and harms their efficiency. Then, assuming data rate's constraint and having $data\_rate = \frac{data\_length}{interarrival\_time}$, we postulate an *interarrival times-data lengths duality* : evading one makes bot vulnerable to the other.

B. Interarrival times

For each packet, we call *interarrival time* the time duration elapsed since last packet arrival. Figure 1 shows interarrival times of chronologically ordered packets within a trace. As displayed, while human trace contains random values, on small sample of consecutive packets, bot exhibits either regular and fast packet arrivals pattern or high and sharp peaks resulting of periods of inactivity (probably due to path finding or character's death). Then, we will consider as bot trace a sample containing only low interarrival time values (according to a threshold) or a sharp peak. Details are given in Section 5.

C. Data lengths

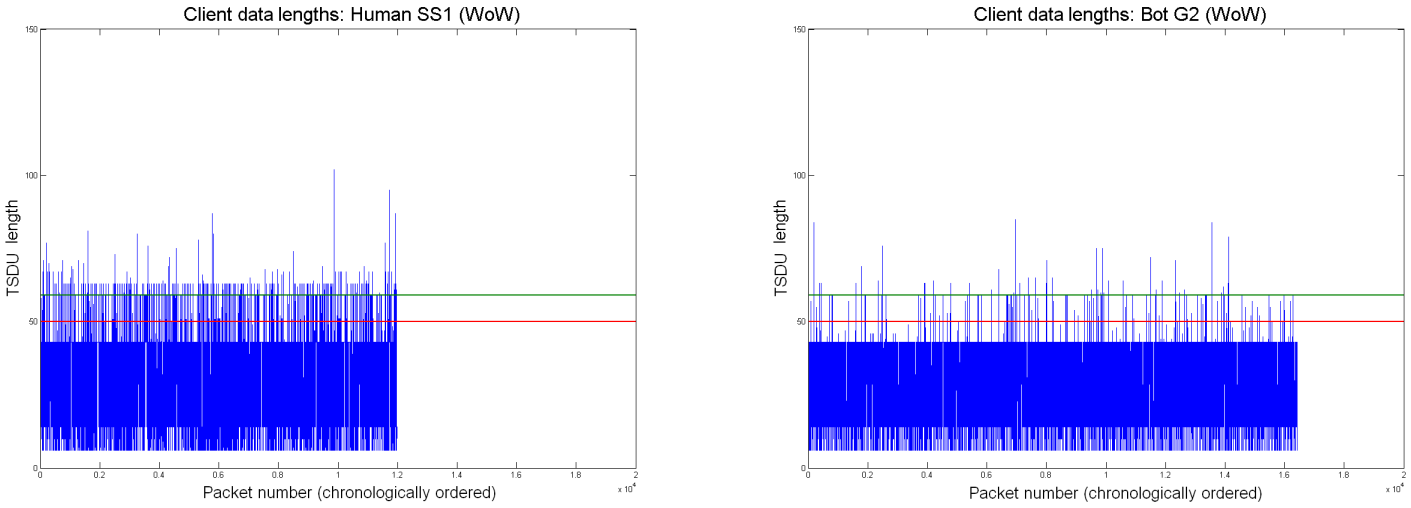
We call *data length* of a packet the size of the corresponding segment's payload (i.e. TSDU's length). Figure 2 shows data lengths of chronologically ordered packets within a trace. As exhibited on the graph, unlike human, bot sends almost no big packets (upper threshold), again affirm our previous rationale. This could be the only test criterion. However we achieve better performance by checking more accurately the data lengths distribution with an additional threshold. Thus, this test checks the ratios of the sample's data length values over each of the thresholds. Details are given in Section 5.

D. Data length autocorrelation

In [14], authors suggest that, in a given data stream, packet size is auto-correlated, which could be adapted to our problem. Moreover, we conjecture that human behavior is generally more random and unpredictable than bot's. As a result, we consider *data length autocorrelation* for another decision test, expecting bot traces to exhibit significant autocorrelation while human traces null one. Figure 3 shows autocorrelation profiles for various traces. Surprisingly, human traces exhibit data length autocorrelation values significantly far from 0. Another unexpected fact is the oscillating shape of Mimic's autocorrelation function. However our study will concentrate on using autocorrelation only as a discriminative feature between bots and humans. Narrowing the scope on the first autocorrelation range, we come up with the following observation: bots' data length autocorrelation is negative, while humans' one is positive. This gives us a threshold based test. Details are again given in Section 5.

V. Online detection scheme

This section contains a detailed description of our online detection scheme.



(a) Human player: trace SS1

(b) Bot player: trace G2

Fig. 2: Client data lengths

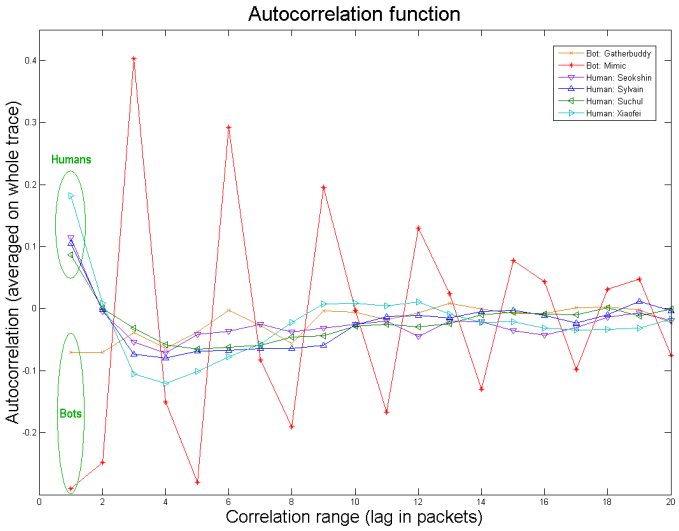


Fig. 3: Autocorrelation function

A. DATA LENGTH AUTOCORRELATION test

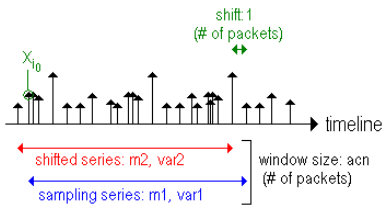


Fig. 4: Principle of autocorrelation online calculation

Due to space limitation, we describe exhaustively the *data length autocorrelation* test, the most elaborate, and will give only brief explanations for other tests and the global detection scheme. We adapt the standard definition of autocorrelation for the sake of on-the-fly calculation on short jumping windows. Figure 4 shows the arrangement of each window. Thus, we calculate it using the sampling series and its shifted counterpart (shift is 1), whose variables are denoted by X and Y, respectively. Then we introduce $m1 = mean(X), m2 = mean(Y), m11 = mean(X^2), m22 = mean(Y^2), m12 = mean(XY)$.

From this, autocorrelation is given by (mean values are easy to calculate on-the-fly):

$$\frac{m12 - m1m2}{sqrt((m11 - (m1)^2)(m22 - (m2)^2))}$$

Algorithm 1 describes the *data length autocorrelation* test. It calculates autocorrelation values on consecutive jumping windows, compare them with a decision threshold and finally performs a majority vote among the consecutive decisions. Its input parameters are the sampling series size *acn*, the number of "voters" *acv* and the decision threshold *acthr*.

B. INTERARRIVAL TIMES test

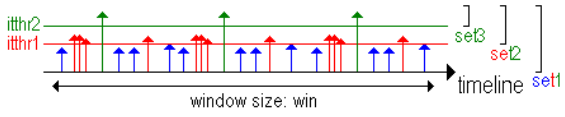


Fig. 5: Principle of INTERARRIVAL TIMES test

This test is twofold: count, within a window (length *win*), the number of values above a threshold *itthr1* and among these values calculate the ratio of values over a second threshold *itthr2*. According to Figure 5, this is equivalent to derive $|set2|$ and $\frac{|set3|}{|set2|}$. The first test checks *regularity* and the second one the *presence of peak*. After having considered several

Algorithm 1 DATA LENGTH AUTOCORRELATION

$vote\_counter, bot\_vote \leftarrow 0$
$m1, m2, m11, m22, m12 \leftarrow 0$
$value\_counter \leftarrow 0$
while new packets arrive **do**
  $x \leftarrow waitNewValue()$ {value stored at packet arrival}
  if $value\_counter \geq 1$ **then**
    $m1 \leftarrow m1 + x$, $m11 \leftarrow m11 + x * x$,
    $m12 \leftarrow m12 + x * pv$
  end if
  $m2 \leftarrow m2 + x$, $m22 \leftarrow m22 + x * x$,
  $pv \leftarrow x$, {storage of shifted value}
  $value\_counter \leftarrow value\_counter + 1$
  if $value\_counter = acn + 1$ **then**
    $m2 \leftarrow m2 - x$, $m2 \leftarrow m2 - x * x$, {correction}
    $m1 \leftarrow m1/n$, $m2 \leftarrow m2/n$, $m11 \leftarrow m11/n$,
    $m22 \leftarrow m22/n$, $m12 \leftarrow m12/n$,
    $d \leftarrow sqrt((m11 - (m1)^2)(m22 - (m2)^2))$,
    $aux \leftarrow (m12 - m1m2)/d$
    if $aux < acthre$ **then**
      $bot\_vote \leftarrow bot\_vote + 1$
    end if
    $vote\_counter \leftarrow vote\_counter + 1$
    if $vote\_counter = acv$ **then**
      if $bot\_vote > acv/2$ **then**
        return BOT
      else
        return HUMAN
      end if
      $vote\_counter, bot\_vote \leftarrow 0$
      $m1, m2, m11, m22, m12 \leftarrow 0$
      $value\_counter \leftarrow 0$
    end if
  end if
end while

possibilities, we found that checking the concentration of very high values out of the "*regularity* zone" is the most efficient technique. Thus, if $|set2|$ is strictly smaller than a threshold *itrt1*, we have *regularity*, and, if $\frac{|set3|}{|set2|}$ is strictly greater than another threshold *itrt2*, we have *presence of peak*. If one of these properties is found for the sample window, the test returns *BOT*. The corresponding online algorithm is trivial.

C. DATA LENGTHS test

This test checks two properties of the sample window (length *win*): the number of values over a threshold *dlthr1* is strictly smaller than a quantity *dlrt1* (*regularity*); the number of values over another threshold *dlthr2* is strictly smaller than another quantity *dlrt2* (*short length*). If the sample window exhibits both *regularity* and *short length* properties, the test returns *BOT*. The corresponding online algorithm is again straightforward.

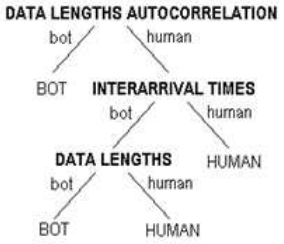


Fig. 6: Global decision scheme's decision tree

D. Global decision scheme

As a global detection scheme, we chose to use a decision tree for its simplicity. The features observed to be more discriminative have higher priority in the tree. Then, at each packet arrival, the different features are updated until *win* packets have been collected. For sake of synchronization, parameters controlling sample window's size have to fulfill the constraint $win = acv(acn + 1)$.

Once all previous tests returned their results, the global scheme uses the decision tree (optimized through experiments) exhibited in Figure 6 to classify the collected trace. It keeps returning real-time decisions while the player is connected to the game server. Its parameters are the sample window's size *win* (reused by *INTERARRIVAL TIMES* and *DATA LENGTHS* tests) as well as those introduced in previous subsections for each specific test. The corresponding online algorithm is trivial and achieves linear time complexity and constant space complexity (according to sample window's size).

VI. PERFORMANCE EVALUATION

To evaluate our proposal, we resort to standard performance metrics: *accuracy* - ratio of well classified traces -, *false alarm rate* - ratio of wrongly classified traces among those classified as bot -, *detection time* - trace length used to return decision, it can be converted into actual time according to packets arrival speed.

TABLE II: Chosen parameters values for tests

Global scheme		**Autocorr.**		**Inter. times**		**Data lengths**	
Param.	*Value*	*Param.*	*Value*	*Param.*	*Value*	*Param.*	*Value*
win	100	acn	19	itthr1	2s	dlthr1	59B
		acv	5	itthr2	6s	dlthr2	50B
		acthr	-0.15	itrt1	1	dlrt1	1
				itrt2	0.3	dlrt2	7

By testing, we came up with the refined parameters' values mentioned in Table II and build our decision tree. However, this training should be done again for a different game. Using a 100 packets long trace - i.e. *detection time* between 18s and 1min40s -, our scheme yields about 86% *accuracy* and around 8% *false alarm rate*. These results are slightly lower than what has been proposed up to now, but not that bad considering the heavy constraints we had to deal with. Moreover, this scheme fulfills the practical requirements for being used as an exhaustive online detection tool. We summarize the main features of our scheme in Table III.

TABLE III: Our proposal's profile

Accuracy	False alarm r.	Type	Detection time	Limitation	Specificity
86.06%	7.74%	traffic based	100 pkts (avg 35s)	little bit weak perf	online detection

VII. Discussions

We end the paper raising a couple of issues related to our work and some others more general related to bot detection. The first one is *generality* . We observed through our experiments that the rationales we proposed (bots send less information than human and *interarrival times-data lengths duality*) are quite strong. Indeed, the tests combination we derived from them leads to a significant performance improvement compared to what which feature yiels individually (about 10% better). In addition, studying Ragnarok traces graciously made public by [1]'s authors (http://mmnet.iis.sinica.edu.tw/content.html?key=ro), we found these rationales to be still meaningful.

It is worthy being noticed that collecting data lengths at client or server side makes no difference for the obtained values. Concerning interarrival times, there might be some difference. However, we use large thresholds for detection (2 and 6s), which are not likely to be much affected by small fluctuations. So, we believe that our proposal may be deployed at server side.

We propose as future directions for our work the design of a *training method* for our proposal (it is indeed required to set parameters manually after having observed data properties at that moment) as well as a study of the *distinction between newbie's and veterans* among human players: are players behaving more like bots as they get more experienced?

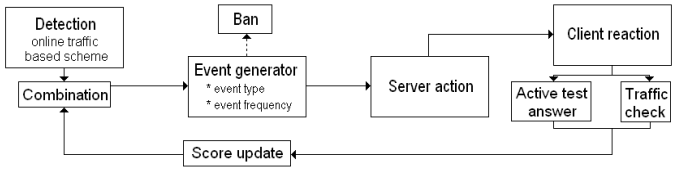


Fig. 7: Systemic bots detection strategy

Finally, we introduce a perspective we have for our work: a *systemic defense strategy*. As we already mentioned in Section 4, our method could be used in combination with more accurate and costly tests, as a ground layer pre-selection tool. We also mentioned that we believe in immediate reaction rather than long scoring process that might finish much later with a ban. Figure 7 gives an insight of our vision. It is also based on iterative scoring (for ban decision), but it uses the fact that Turing tests are active (i.e. they disturb the player). A suspicious player could then be sent more frequent tests. In case of bots, it means either greater chances to be caught or presence of the human behind the machine (so bothersome). In addition, coupling our test with a CAPTCHA to catch traffic patterns' short variations (our detection time is small), we even might have some success against semi-autonomous bots, who call back humans to answer Turing tests. Finally, the idea is to trigger tests for higher accuracy and increase their frequency for bigger annoyance.

VIII. Conclusion

In this work, we have drawn a general picture of the related research field which showed that the bot detection is not yet apprehended as a sub part of the anti-bot defense. Then, we have propose an online bot detection scheme based on traffic pattern analysis exhibiting low time and space complexity and candidate for ground layer exhaustive check, which could be a component of a bigger anti-bot defense system.

Acknowledgment

This work was supported by NAP of Korea Research Council of Fundamental Science and Technology and the ITRC support program [NIPA-2010-C1090-1011-0004] of MKE/NIPA. The ICT at Seoul National University provided research facilities for this study.

References

[1] K.-T. Chen, J.-W. Jiang, P. Huang, H.-H. Chu, C.-L. Lei and W.-C. Chen, *Identifying MMORGP Bots: A Traffic Analysis Approach*, In EURASIP Journal on Advances in Signal Processing, Volume 2009.

[2] Y. Rui and Z. Liu, *ARTiFACIAL: Automated Reverse Turing test using FACIAL features*, In Proceedings of the 11th ACM international conference on Multimedia, 2003.

[3] L. v. Ahn, M. Blum, N. J. Hopper, and J. Langford, *CAPTCHA: Using Hard AI Problems For Security*, In Proceedings of Eurocrypt 2003.

[4] R. Gossweiler, M. Kamvar, and S. Baluja, *Whats Up CAPTCHA? A CAPTCHA Based on Image Orientation*, In Proceedings of the 18th international conference on World wide web, 2009.

[5] R. V. Yampolskiy and V. Govindaraju, *Embedded Noninteractive Continuous Bot Detection*, In Computers in Entertainment Volume 5, 2008.

[6] P. Golle and N. Ducheneaut, *Keeping Bots out of Online Games*, In Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology, 2005.

[7] P. Hingston, *A Turing Test for Computer Game Bots*, IEEE Transactions on Computational Intelligence and AI in Games, 2009.

[8] H. Kim, S. Hong and J. Kim, *Detection of Auto Programs for MMORPGs*, In Proceedings of AI 2005: Advances in Artificial Intelligence.

[9] S. Mitterhofer, C. Platzer, C. Kruegel and E. Kirda, *Server-Side Bot Detection in Massively Multiplayer Online Games*, In IEEE Security and Privacy Volume 7, 2009.

[10] K.-T. Chen, H.-K. Kenneth Pao and H.-Ch. Chang, *Game Bot Identification Based on Manifold Learning*, In Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, 2008.

[11] K.-T. Chen, A. Liao, H.-K. Kenneth Pao and H.-H. Chu, *Game Bot Detection Based on Avatar Trajectory*, In Proceedings of the 7th International Conference on Entertainment Computing, 2008.

[12] K.-T. Chen, L.-W. Hong, *User IdentificationbasedonGame-PlayActivityPatterns*, In Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games, 2007.

[13] R. Thawonmas, Y. Kashifuji and K.-T. Chen, *Detection of MMORPG Bots Based on Behavior Analysis*, In Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology.

[14] P. Branch and G. Armitage, *Measuring the auto-correlation of server to client traffic in First Person Shooter games*, In Australian Telecommunication Networks and Application Conference, 2006.

[15] Wikipedia, *Turing test*, http://en.wikipedia.org/wiki/Turing_test.