Identifying MMORPG Bots: A Traffic Analysis Approach*

Kuan-Ta Chen[12], Jhih-Wei Jiang[3], Polly Huang[14], Hao-Hua Chu[34],
Chin-Laung Lei[14], and Wen-Chin Chen[34]

[1]Department of Electrical Engineering, National Taiwan University
[2]Institute of Information Science, Academia Sinica
[3]Department of Computer Science and Information Engineering, National Taiwan University
[4]Graduate Institute of Networking and Multimedia, National Taiwan University

ABSTRACT

MMORPGs have become extremely popular among network gamers. Despite their success, one of MMORPG's greatest challenges is the increasing use of game bots, i.e., auto-playing game clients. The use of game bots is considered unsportsmanlike and is therefore forbidden. To keep games in order, game police, played by actual human players, often patrol game zones and question suspicious players. This practice, however, is labor-intensive and ineffective. To address this problem, we analyze the traffic generated by human players vs. game bots and propose solutions to automatically identify game bots.

Taking *Ragnarok Online*, one of the most popular MMOGs, as our subject, we study the traffic generated by mainstream game bots and human players. We find that their traffic is distinguishable by: 1) the regularity in the release time of client commands, 2) the trend and magnitude of traffic burstiness in multiple time scales, and 3) the sensitivity to network conditions. We propose four strategies and two integrated schemes to identify bots. For our data sets, the conservative scheme completely avoids making false accusations against bona fide players, while the progressive scheme tracks game bots down more aggressively. Finally, we show that the proposed methods are generalizable to other games and robust against counter-measures from bot developers.

Keywords

Game Bot, Online Games, Traffic Burstiness

1. INTRODUCTION

MMORPGs (Massive Multiplayer Online Role Playing Games) have become extremely popular among network gamers, and now attract millions of users to play in an evolving virtual world simultaneously over the Internet. The number of active player subscriptions [12] doubled between July 2004 and June 2005 to a 500-million player base. Despite their success, one of MMORPG's greatest challenges to maintain the subscription base is the use of game bots[1].

A game bot, usually game-specific, is an automated program that can perform many tasks in place of gamers. Since bots never get tired, bot users can improperly reap rewards with less time investment than legitimate players. As this undermines the delicate balance of the game world, bots are usually forbidden in games. However, identifying whether or not a character is controlled by a bot is difficult, since a bot-controlled character *obeys the game rules completely.* The state of the practice of bot identification is to *manually* launch a dialogue with a suspect character, as a bot cannot talk like a human. But this method leads to a significant administrative burden. To the best of our knowledge, this work is the first to address automatic bot identification techniques using traffic analysis.

Taking *Ragnarok Online*[2], one of the most popular MMOGs in the world, as a case study, we analyze the traffic of mainstream game bots and human players with different network settings. We find that traffic due to bots vs. human players is *distinguishable* in various respects, such as the regularity and patterns in client response times, i.e., the release time of client commands relative to the arrival time of the most recent server packet, the trend and magnitude of traffic burstiness in multiple time scales, and the sensitivity to network conditions. Based on the above findings, we derive four strategies *to determine whether or not a given traffic stream corresponds to a game session played by a game bot.* Using proper combinations, we propose two integrated schemes, one conservative and one progressive, to automatically identify game bots. Our evaluation shows that the conservative scheme reduces the false positive rate to zero and achieves 90% accuracy in identifying bots. Meanwhile, the progressive scheme yields a false negative rate of less than 1% and maintains 95% accuracy. The proposed methods are later shown generalizable to other bots and games and robust against counter-measures from bot developers.

The remainder of this paper is organized as follows. Section 2 describes related works. Section 3 provides a brief introduction of the game *Ragnarok Online* and an assessment on the current status of game bots. Section 4 discusses the

*This work is supported in part by the Ministry of Economic Affairs under the Grant No. 94-EC-17-A-02-S1-049, and by the Taiwan Information Security Center (TWISC), National Science Council under the Grants No. NSC 94-3114-P-001-001Y and NSC 94-3114-P-011-001.

ACE'06, June, 2006 Hollywood, California, USA

[1]http://en.wikipedia.org/wiki/MMORPG#Bots
[2]http://iro.ragnarokonline.com/

Figure 1: A screen shot of the game *Ragnarok Online*. On the screen about eight characters are holding a sorcery ritual. Figure courtesy of http://www.Ragnarok.co.kr.

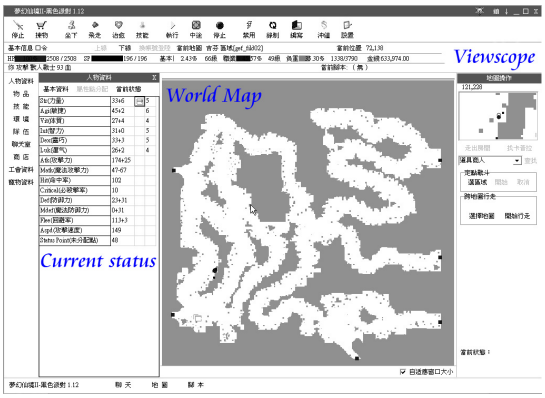

Figure 2: A screen shot of the *DreamRO* bot.

trace collection. Section 5 analyzes the discrepancies between traces for bots and human players. Then, based on our findings, we propose four bot identification strategies in Section 6. Section 7 evaluates the performance of the proposed schemes, and discusses their practical use in real business operations. We discuss the generality and robustness of the schemes in Section 8. Finally, Section 9 draws our conclusion.

2. RELATED WORK

While cheating is regarded as a crucial challenge to the design of online games, a great deal of effort has been devoted to cheat prevention [1, 4, 5, 13]. Game cheats often exploit loopholes in game rules or specific implementations, so researchers attempt to guarantee the correctness of game systems by, for example, runtime verification of transaction atomicity [5]. However, the correctness proof approach does not apply to bot detection problems because game bots *obey the game rules completely.*

3. RAGNAROK ONLINE AND THE BOTS

Ragnarok Online, one of the most popular MMORPGs worldwide, was created by the Gravity Corporation in 2002. According to MMOGCHART [12], *"Ragnarok Online is supposedly the second biggest MMOG in South Korea, with well over 2 million subscribers. ... Recently they claimed 17 million worldwide with over 700,000 in North America, ..."* One reason for its popularity is the well-rendered comic style that perfectly integrates 2D and 3D graphics. The screen shot of *Ragnarok Online* in Fig. 1 shows eight characters holding a sorcery ritual. The game design encourages players to get involved with other characters and the community, which is another reason for *Ragnarok Online*'s popularity.

The core aspects of most MMORPGs are more or less standard, e.g., training characters, obtaining better equipment, and completing various quests, which usually involve fighting with monsters. Characters gradually become stronger and better equipped by gaining experience points and by accumulating loot from combat. However, repeated combat is time-consuming and can become somewhat routine and boring; thus, some players seek to set up scripts (also known as macros or bots) that can automatically and repeatedly perform assigned tasks without human involvement. Given that bots never get tired, bot users can reap huge rewards without the time investment made by other honest players.

From the view point of business operation, bots erode the balance and order of the game world, as bot users can monopolize scarce resources by unleashing the indefatigable power of bots. Although companies try to prevent the use of game bots, automatic bot detection mechanisms are not currently available; thus, bot-controlled characters *can only be identified manually through human intelligence.* That is, game masters try to open online dialogues with suspicious characters; the game masters can decide if they are actually bot-controlled or human-controlled based on their responses. However, this method is very inefficient given millions of online players, and leads to a significant administrative burden. The biggest restriction is that the method *can only be applied on suspected characters.* As the problem of players cheating with game bots becomes rampant and serious, we believe the development of automatic bot identification techniques for MMORPGs is urgent.

We surveyed publicly-available game bots for *Ragnarok Online*, and found that though more than a dozen bots are available, most of them derive from the well-known *Kore* project[3]. *Kore* is a console-based, platform-independent, and open-source bot program written in Perl and C. It could be described as the ancestor of *Ragnarok Online* bots, since many popular bots, e.g., *KoreC*, *X-Kore*, *modKore*, *Solos Kore*, *wasu*, *Erok*, *iKore*, and *VisualKore*, have been developed from it. Similar to Kore, we found that another bot series, DreamRO[4] and its derivatives, is also very popular in China and Taiwan.

Both *Kore* and *DreamRO* are standalone bots, i.e., they can communicate directly with game servers without the official game clients. Their actions are script-based, covering almost every action available in the game client. When a bot program is running, it continuously reports the latest information and the current status of the game, e.g., a character's location, the current action, the "hit" point, and information about nearby monsters. Both *Kore* and *DreamRO* allow users to give commands anytime, regardless of the prearranged actions by scripts, i.e., the bots are both

[3]Kore, `http://sourceforge.net/projects/kore/`
[4]DreamRO, `http://www.ayxz.com/soft/1805.htm`

Table 1: Game Traffic Traces (206 Hours and 3 Million Packets in Total)

Category	Player	ID	Network*	Period	# Conn	Pkt	Bytes	Pkt Rate‡	Avg RTT	Loss
Human Player	Gino	*A1*	HiNet	1.8 hr	12	51,823	3.5 MB	0.9 / 3.9 pkt/s	82.0 ms	0.03%
		A2	2M/512Kbps	5.6 hr	14	147,814	10.5 MB	0.8 / 3.4 pkt/s	95.4 ms	0.03%
	Kiya	*B1*	APOL 2M/512Kbps	0.4 hr	45	15,228	1.0 MB	1.2 / 4.5 pkt/s	81.6 ms	0.01%
		B2		2.3 hr	108	59,247	3.8 MB	1.1 / 3.3 pkt/s	108.8 ms	0.12%
		B3		2.1 hr	189	47,721	3.2 MB	0.9 / 2.8 pkt/s	125.5 ms	0.23%
		B4		5.0 hr	326	129,177	8.4 MB	1.1 / 3.3 pkt/s	109.8 ms	0.09%
	Kuan-Ta	*C1*	ASNET†	0.8 hr	2	9,681	0.6 MB	0.8 / 1.4 pkt/s	191.8 ms	1.73%
	Jhih-Wei	*D1*	TANET	2.4 hr	28	48,617	3.2 MB	0.8 / 2.6 pkt/s	45.1 ms	0.01%
Bot	Kore	*K1*	TANET	13.4 hr	104	245,709	13.6 MB	0.7 / 2.3 pkt/s	33.0 ms	0.01%
		K2		26.5 hr	306	479,374	30.4 MB	1.0 / 2.1 pkt/s	45.6 ms	0.04%
		K3		32.7 hr	37	271,416	13.3 MB	0.6 / 0.7 pkt/s	96.5 ms	0.004%
		K4	ETWEBS-TW	13.0 hr	38	225,528	11.5 MB	0.9 / 2.0 pkt/s	65.7 ms	0.01%
		K5		5.7 hr	31	110,883	6.0 MB	1.1 / 2.1 pkt/s	90.6 ms	0.20%
	DreamRO	*R1*	TANET	3.0 hr	7	46,381	2.6 MB	0.9 / 1.7 pkt/s	83.4 ms	0.03%
		R2		4.8 hr	21	77,675	4.4 MB	0.9 / 1.9 pkt/s	65.2 ms	0.02%
		R3		42.3 hr	42	652,877	34.1 MB	0.8 / 1.8 pkt/s	85.3 ms	0.05%
		R4	ETWEBS-TW	11.2 hr	77	320,686	25.1 MB	1.7 / 3.5 pkt/s	85.2 ms	0.05%
		R5		23.1 hr	176	672,325	53.3 MB	1.7 / 3.6 pkt/s	79.4 ms	0.16%
		R6		10.5 hr	36	209,347	13.1 MB	1.0 / 2.4 pkt/s	87.7 ms	0.05%
Total	2 B / 4 P	19		206.6 hr	1,599	3,821,509	241.6 MB			

* This column lists network names looked up using WHOIS service.
† Access link bandwidth: ASNET (2M/512Kbps), ETWEBS-TW (2M/256Kbps), and TANET (100Mbps)
‡ Packet rate column format is "client data packet rate / server data packet rate," i.e., pure TCP ack packets do not count.

script-based and interactive. Fig. 2 shows the main window of a running *DreamRO* bot, in which the character is chasing a monster. In the picture, the world map (large map) is the current map where the character is located, and the view scope (small map) indicates there is a monster nearby. The status panel lists the current status of the character.

4. TRACE COLLECTION

To develop bot identification techniques based on discrepancies in traffic patterns, we acquired a number of *Ragnarok Online* game traces for both popular bot series and for human players. For brevity, we use "players" to denote human players hereafter. To make trace collection tractable, we chose a bot program to represent each series. *KoreC*, the Chinese edition of *Kore*, was selected as the representative of the *Kore* series, and *DreamRO* was chosen to represent the *DreamRO* series. Although only one bot was analyzed for each series, we believe bots in the same series behave similarly based on the same kernel implementation.

We collected a total of 19 game traces at the client side, i.e., the traffic monitor was attached to the same LAN as the game client. In order to ensure heterogeneity among the limited number of traces, we intentionally incorporated combinations of controllable factors into the trace collection. From a networking perspective, both bot and player traces contained fast (campus network) and slow (residential connection) access links, and the network media ranged from Fast Ethernet to ADSL and cable-modem. In terms of user behavior, the human players were diverse in their choice of characters and game playing proficiency: among the four players, Gino and Kiya were experienced—both of them had played *Ragnarok Online* for more than one year—and their characters were high-level (> level 60), well-equipped, and highly-skilled. On the other hand, Kuan-Ta and Jhih-Wei were newcomers to *Ragnarok Online*, and their characters were low- to middle-level (level 5 and level 40 respectively) without advanced skills or powerful weapons. The scripts we used for the two bots are commonly available among the *Ragnarok Online* community. Their actions are set to the most common "kill, loot, and trade" cycles. In other words, at the start, the bot will go to a selected area, where there are many monsters, and proactively pursue and attack the nearest monster. After the monster is killed, the bot will pick up the loot, and turn to another monster. The process will continue until the backpack is full of loot. At that time the bot will return to a marketplace to sell the gathered loot, and then restart the cycle. As in the human player case, we purposely ran the bots with characters of different levels of proficiency and professions.

The collected game traces are summarized in Table 1. For brevity, we denote the four players as *A*–*D*, Kore as *K*, and DreamRO as *R*. Traces from the same bot/player are coded by a unique digit following the bot/player's code. The average client packet rate is an indicator of *player activity*, since each player command is conveyed by a client data packet. On the other hand, the average server packet rate is an indicator of *interaction*, i.e., how popular and active the area where the character resides is, as server packets convey nearby characters' activities [3]. Note that the average packet rate is roughly the same for the same bot/player with the same network setting, which may be seen as a "signature" of the game playing behavior of a certain bot/player. The behavior of our selected human players can be shown to be heterogeneous by these two metrics. Also, the average round trip times (RTT) and packet loss rate statistics manifest the heterogeneity of the network QoS the traced sessions experienced.

5. ANALYSIS OF TRAFFIC PATTERNS

From a traffic analysis perspective, the most intuitive discrimination between bots and players is probably the *release timing of client commands*. For human players, client com-

mands, e.g., approach another character, attack a nearby monster, or cast healing magic, are triggered by keyboard strokes or mouse clicks. On the other hand, for game bots, triggering client commands is decided by the decision engine in the bot program. Thus, the design of *when to issue the next command* is important to us because it leads to major discrepancies in traffic patterns between different bot series, and between bots and human players.

Following an analysis of the release timing of client commands from bots, we find that the release of commands, for both *Kore* and *DreamRO*, depends on the following events: 1) *timer expiration*, and 2) *server data packet arrivals*. The use of periodic timers is intuitive and reasonable, since many actions in a game are *iterative in nature*, e.g., continuous slash until an enemy is defeated. On the other hand, since server data packets carry the latest status of the character and environment, e.g., the current life point of the character, the movement of nearby monsters, and whether the last slash hits the enemy, bots often react to server data packets by issuing new commands. For example, to pursue a fleeing enemy, a bot can continuously issue movement commands whenever it is aware of the latest location of the enemy from the server data packets.

In the following, we analyze the traffic traces of game bots and human players, and search for distinctive traffic patterns exhibited by bots, but not by players, and vice versa. The analysis of traffic patterns comprises three aspects. First, we examine the timing of client commands relative to the arrival time of the most recent server data packet. We then observe the traffic burstiness of the packet traces. Lastly, we identify particular patterns of human behavior caused by sensitivity to network conditions, which, of course, is not possessed by game bots.

For the sake of brevity, we use "client packets" to denote data packets sent out by game clients, excluding pure TCP acknowledgement packets; "server packets" are similarly defined. In addition, since we concentrate primarily on client packets, the term "packets" denotes "client packets," unless otherwise specified.

5.1 Command Timing

We start with the definition of "client response time" as the *time difference between a client packet departure time and the most recent server packet arrival time*, if no other client packets intervene; otherwise, the metric is undefined. Since we do not consider the corresponding server response time, for brevity, we use "response time" to denote client response time hereafter. By the above definition, for each trace, we compute the response times for those client packets that immediately follow a server packet. As an initial assessment of whether the response times of bot traces differ significantly from those of player traces, cumulative distribution functions of response times of less than 0.1 seconds for four traces are plotted in Fig. 3. In the figure, except for an initial rise for *A1*, the two player traces, *A1* and *B2*, are similar in that their response times less than 0.1 seconds increase smoothly, i.e., they are almost uniformly distributed. On the other hand, bot traces reveal different patterns: the CDF of *Kore1* is a zigzag-type, i.e., the response times are clustered around certain intervals, while that of *DreamRO2* has a strong mode with very small response times. In the following, we discuss these two properties of bot traces, i.e., strong modes and zigzag CDF, in more depth.

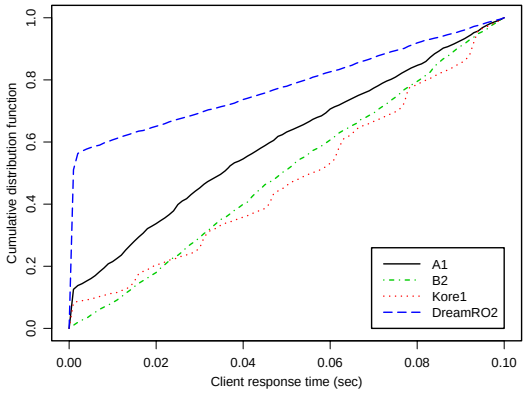

Figure 3: CDF of client response times

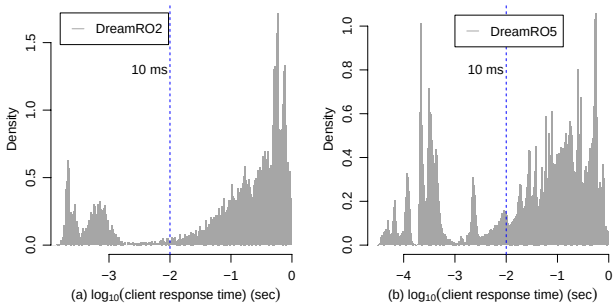

Figure 4: Histograms of client response times in two *DreamRO* traces. More than one peak is formed at time scales smaller than 10 ms. Peaks do not occur in player traces or *Kore* traces.

5.1.1 Prompt Response

Among all game traces, only those of *DreamRO* possess a considerable number of short response times, e.g., ≤ 10 ms, which we call *prompt responses*. These responses are frequent enough and clustered so that more than one peak is formed in the corresponding histogram, as shown in Fig. 4. Note that to distinguish peaks clearly we take a logarithm of the response time. The prompt response manifests that the design of *DreamRO* often issues client commands immediately upon the receipt of server packets, while *Kore* employs a more sophisticated command timing mechanism.

5.1.2 Regularity in Response Times

Although prompt responses are not present in the traces of *Kore*, it still relies on server packet arrival events to schedule the release of client commands. In Fig. 5, which depicts histograms of response times shorter than 0.5 second, both bot traces show *spiky* densities, while player traces do not present any visible patterns. These plots indicate that both *Kore* and *DreamRO* schedule their client commands by an intentional delay time following receipt of a server packet. In the histogram, if the bin width is small enough, the distance between *spikes* will reflect the smallest scheduling unit of the command departure times; according to our traces, the value is set to 15 ms for both *Kore* and *DreamRO*. In Section 6.1, we will propose a bot detection scheme based on

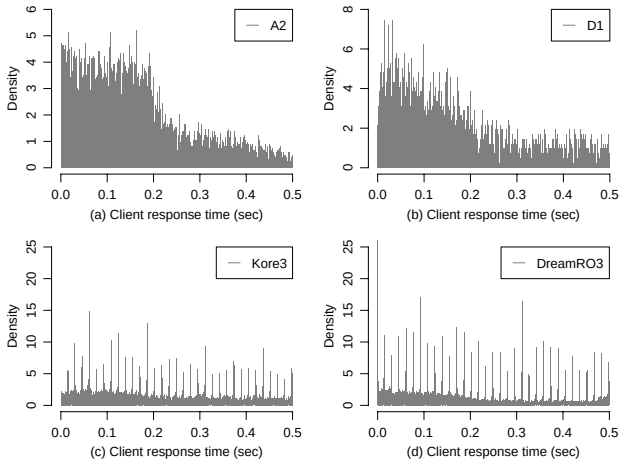

Figure 5: Histogram of client response times shorter than 0.5 seconds

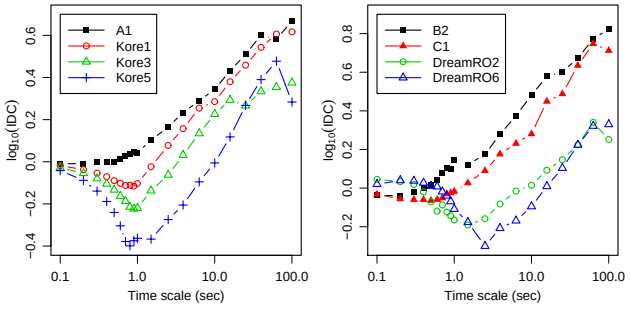

Figure 6: IDC plots for different traces. Player traces are represented by filled symbols; bot traces are represented by unfilled symbols. Note the trend of IDC for bot traces has a "dip" around 1 second.

the prompt responses and the regularity in response times identified above.

5.2 Traffic Burstiness

Traffic burstiness, i.e., the variability of traffic volume, or the packet count sent in successive periods, is an indicator of how traffic fluctuates over time. While traffic burstiness is commonly related to the scaling property of a traffic stream, we use it to assess how *bursty* the bot traffic is. Our hypothesis is that a bot, by virtue of its periodicity, should exhibit smoother traffic compared with player traffic.

Like all other programs, a bot program must have a *main loop*, where an iteration of the loop corresponds to a minimum unit of operation, e.g., issuing a command for a character, or processing a server packet. The rationale behind multi-time scale burstiness analysis is that, assuming each iteration (of the main loop) takes approximately the same amount of time, and in each iteration the game bot sends out roughly the same number of packets, then *traffic burstiness will be lowest at the time scale equal to the amount of time needed for each iteration of the main loop.*

To measure traffic burstiness in multiple time scales, we use the index of dispersion for counts (IDC). The IDC at time scale t is defined as the variance in the number of arrivals in an interval of time t divided by the mean number of arrivals in t [8], that is,

$$I_t = \frac{\mathrm{Var}(N_t)}{E(N_t)}, \quad (1)$$

where N_t indicates the number of arrivals in an interval of time t. The IDC is thus defined so that, for a Poisson process, the value of the IDC is 1 for all t.

The IDCs for selected game traces are plotted in Fig. 6. We make two observations from the plots: 1) bot traffic is smoother than player traffic, but it is hard to define a threshold for the burstiness magnitude; and 2) all bot traces support our hypothesis that they have the lowest burstiness at time scales around 0.5–2 seconds. In other words, the burstiness initially exhibits a "falling trend" when the time scales are small, but after a certain time scale with the lowest burstiness, a "rising trend" will appear. In contrast, the burstiness trends of most player traces monotonically increase in time scales > 1 second. We will exploit these patterns to develop a bot identification scheme in Section 6.2.

Another aspect we investigate is the *magnitude of traffic burstiness*. Though we cannot judge how smooth a traffic process is simply by the absolute value of IDC measures; in our case, we can take the IDC of server packet arrivals as the baseline, and obtain the *relative smoothness of client packet arrivals*. The rationale behind the comparison is that, even if the client traffic is extremely different, game servers still treat all clients *equally*, i.e., the burstiness of server traffic processes, especially in larger time scales, should be similar regardless of the client type. For comparison, we first normalize the server packet arrival process so that it has the same mean rate as client packet arrivals. Then we define the *cross-point* as *the minimum time scale where the burstiness of the client traffic is lower than that of the corresponding server traffic* to determine the relative smoothness of the client traffic. Fig. 7 shows the burstiness comparison for selected traces, where the dashed vertical line denotes the *cross-point*. According to the plots, while both client types have server traffic of similar burstiness trend and magnitude, bot traces have cross-points at lower time scales (< 1 second) than player traces due to their relatively smoother client traffic. This property will be further exploited to identify bots in Section 6.3.

5.3 Sensitivity to Network Conditions

The last aspect we consider is the *subconscious human reactions to network conditions* embedded in traffic traces, which is considerably different from previous approaches. Our finding is that, *human players adapt to the game pace involuntarily*. While game pace is built into network game clients, it is inevitably affected by network conditions as the latest information about other characters and the environment is conveyed by server packets. In short, the transit delay of server packets influences the pace of game playing and therefore a player's pace. To evaluate how network delay affects a player's pace, samples of round trip times (RTT) as well as the average packet rate within the next second following each RTT sample are computed. The plots describing the relationship between average packet rates and RTTs, where the RTTs are grouped in units of 10 ms, are depicted in Fig. 8.

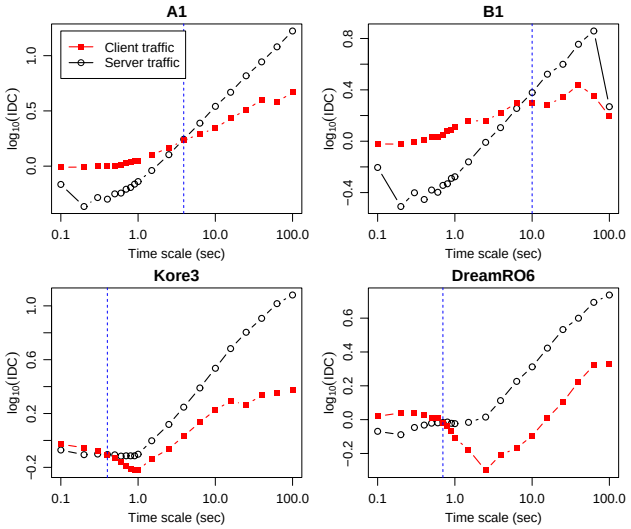

Figure 7: IDC magnitude comparison: The IDC of client packet arrivals versus the IDC of server packet arrivals.

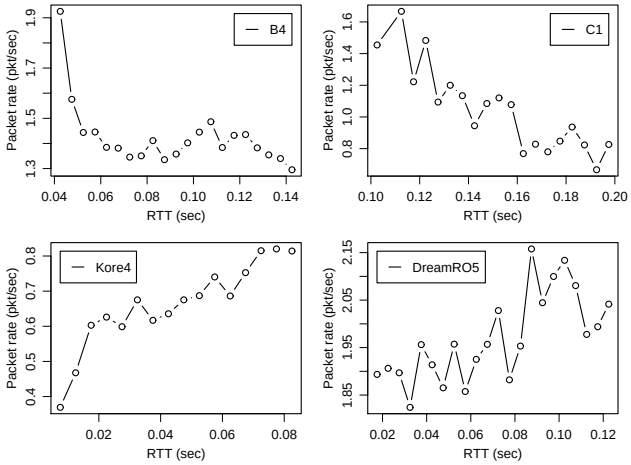

Figure 8: Average packet rates vs. round trip times plot. The figures exhibit a downward trend for player traces, and a upward trend for bot traces.

First we analyze the player traces shown in Figures 8(a) and (b). The trend is clearly *downward*. For slower game pace, which is caused by severely delayed server packets, human players unconsciously slow down their keyboard and mouse actions so that the corresponding packet rate is lower. Figures 8(c) and (d) indicate that the same phenomenon does not appear in bot traffic: both the *Kore* and *DreamRO* traces show a *upward* trend in the relationship of the packet rate versus RTT. Since bots have their own pacing schemes (certain frequencies by timers), they are not *paced* by server packets like human players. One possible explanation of the bots' upward trend (instead of no trend) is that, for a server packet that arrives late, bots issue more commands which are accumulated before the arrival of that server packet. The pacing property exhibited by human players will be further exploited in Section 6.4 as a means of distinguishing bots from human players.

6. PROPOSED DETECTION METHODS

In this section, we propose four decision schemes for the bot identification problem. A decision scheme for a given packet trace will output a dichotomous answer—true or false—to indicate whether or not the trace corresponds to a game session played by a game bot. In the following, we present our methods and the preliminary results. A more complete performance evaluation of these methods is provided in the next section.

6.1 Command Timing

Our first method, *command timing*, relies on two properties related to the client response time following receipt of server packets, which was described in Section 5.1. In this scheme, we simultaneously apply two tests: 1) whether *multiple peaks* exist in the histogram of client response times less than 10 ms; and 2) whether *regularity* exists in response times less than one second. The scheme returns true if either test is true, and false otherwise.

6.1.1 Multimodality Test

To detect the multimodality of response times less than 10 ms, we use the Dip test [9, 10], which is designed to test unimodality. On the response time histogram, we first find all local peaks and troughs; then, for each candidate mode, which is determined by two troughs with at least one peak in between, we apply the unimodality test for the candidate, i.e., if the Dip statistic is significant at level 0.05. The multimodality test is passed if and only if we can successfully identify two or more modes for response times smaller than 10 ms.

6.1.2 Regularity Test

As discussed in Section 5.1.2, client response times in bot traces show highly regular patterns in the form of response times clustered in multiples of a certain value (cf. Fig. 5). To check the existence of such regularity, we take the histogram of response times as a spatial series, and check the existence of frequency components in that series. For a histogram with n bins, we apply a Fourier transform on its ordinates by:

$$I(f) = n\,|d(f)|^2\,,$$

where $d(f)$ is the discrete Fourier transform of that series at frequency f, and $I(f)$ is known as the periodogram. In Fig. 9, the corresponding periodograms of histograms in Fig. 5 are depicted. The strong spikes in the periodograms for bot traces show clear evidence of regularity in the response times.

We adopt the Fisher's test to judge whether periodicity exists in periodograms, which is equivalent to the existence of regularity in the response times. Fisher [6] proposed a test of the significance of the largest peak in the periodogram, which is used to determine if the prominent frequency component is "strong enough." The test statistic is the ratio of the largest periodogram ordinate at the Fourier frequencies to the sum of the ordinates. Fuller [7] proposed an equivalent statistic, which is the ratio of the largest ordinate to the average of the ordinates. While the null hypothesis is that the data consists of white noise, Section 6.8 in [2] suggests

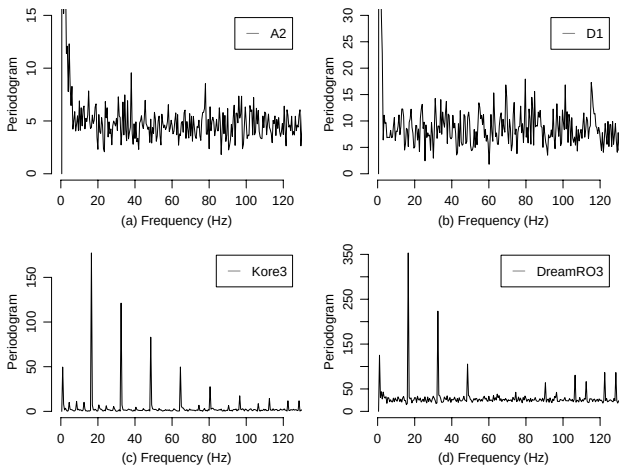

Figure 9: Periodograms of corresponding histograms (of client response times) in Fig. 5. Note strong frequency components exist in bot traces.

that, even if the Gaussian assumption is not satisfied, the theory should continue to provide a useful approximation. Suppose $I_1, I_2, \ldots, I_m$ are periodogram ordinates, then by the null hypothesis, $I_1, I_2, \ldots, I_m$ are independent and exponentially distributed with mean σ^2; that is,

$$P(I_j/\sigma^2) = 1 - e^{-x}, x \geq 0, j = 1, 2, \ldots, m.$$

Let $X_m = max(I_1, I_2, \ldots, I_m)$, $Y_m = \sum_{i=1}^{m} I_i$, and Fuller's test statistic be defined as $\xi_m = X_m/(Y_m/m)$. The significance value for Fuller's test is obtained by

$$P(\xi_m \leq \xi) \approx \exp(-me^{-\xi}).$$

Using this method, we test the regularity in response times. A trace is declared as corresponding to a bot if Fuller's statistic is significant at 0.01.

6.2 Trend of Traffic Burstiness

We now turn to the second identification method. In this method, we use the property that bot traffic will exhibit the lowest burstiness at a time scale approximately equal to the iteration time of its main loop. (cf. Section 5.2)

To check whether the burstiness initially exhibits a falling trend followed by a rising trend, we use the Mann-Kendall correlation test [11] to detect the trend of a pair of series. The nonparametric Mann-Kendall test is expected to be robust to outliers, because its statistics are based on the ranks of variables, not directly on their values. Given IDC ordinates, $\{I_t\}$, where $t > 0.1$ is the corresponding time scale, this scheme comprises two sub-tests: 1) Whether (t, I_t) exhibits a significant falling trend followed by a significant rising trend (both at a significance level 0.05), and whether both trends can be detected in time scales smaller than 10 seconds. 2) Whether any time scale $t' > 10$ exists so that $\{(t, I_t), t < t'\}$ exhibits no significant trend, or a significantly negative trend. The scheme outputs true if either test is true; otherwise, it outputs false.

6.3 Magnitude of Traffic Burstiness

As exemplified in Fig. 7, the burstiness of client traffic is comparable to that of normalized server traffic. Also, recall that we define the "cross-point" as a metric of how smooth the client traffic is. The method based on the magnitude of traffic burstiness is as follows. For a given packet trace, we compute the IDCs for client and server traffic, and search for the cross-point. If the cross-point does not exist, i.e., the client traffic is always more bursty than the server traffic, it is set to the maximum time scale we use, which is 100 seconds. By observation, we set a threshold at 10 seconds so that a trace is said to correspond to a game bot if the cross-point is smaller than 10 seconds, and to a human player otherwise. The decisions are exact in most cases for player traces; however, some bot traces are classified as human players, especially for *Kore* traces. This suggests that the burstiness of server traffic may not be a very good baseline, since it depends on the region where the character resides and the activity of nearby characters. Nevertheless, this method merits our attention as it yields the minimum false positive rate, i.e., the number of times of a player is misjudged as a bot.

6.4 Reaction to Network Conditions

In Section 5.3, we investigated the relationship between round trip times and the corresponding packet rate; that is, human players subconsciously adapt to network delay, and therefore a *negative* correlation exists between the RTT and the packet rate. In contrast, the RTT and the corresponding packet rate are *not correlated or positively correlated* for bot traces. The last scheme is based on the above properties. For a given trace, we first take the RTT samples and the corresponding packet rates within the next second of the RTT sampling times. Then we group these samples by RTT with group size 10 ms such that a series, $\{(RTT_i, N_i), i \geq 1\}$, is formed, where RTT_i is the center point of group i and N_i is the average packet rate of samples in group i. We use the Mann-Kendall test to detect the trend of packet rates versus RTT. The method reports a bot if the τ statistic is statistically greater or equal to zero, and a human player otherwise.

7. PERFORMANCE EVALUATION

In this section, we evaluate proposed bot identification schemes. For each scheme, three metrics are evaluated: the *correct rate*, the ratio the client type of a trace is correctly determined; the *false positive rate*, the ratio a player is judged as a bot; and the *false negative rate*, the ratio a bot is judged as a human player. In addition, we are concerned about the sensitivity of the input size, i.e., *how long a traffic stream must be to enable correct identification*. Thus, the performance metrics are computed on a segment basis by dividing the traces into segments of a given size.

The evaluation results reveal that the first two methods, *command timing* and *burstiness trend*, work rather well, as shown in Fig. 10. Specifically, both methods can achieve correct decision rates higher than 95% and false negative rates lower than 5%, given an input size larger than $10,000$ packets. From the viewpoint of business operations, a good bot detection method should minimize the false positive rate while yielding a high correct decision rate. This is because misjudging a human player as a bot would annoy legitimate players, while misjudging a bot (as a human player) would be relatively acceptable. By this rule, the *burstiness magnitude* method is good, since it always achieves low false positive rates ($< 5\%$), and yields a moderate correct deci-

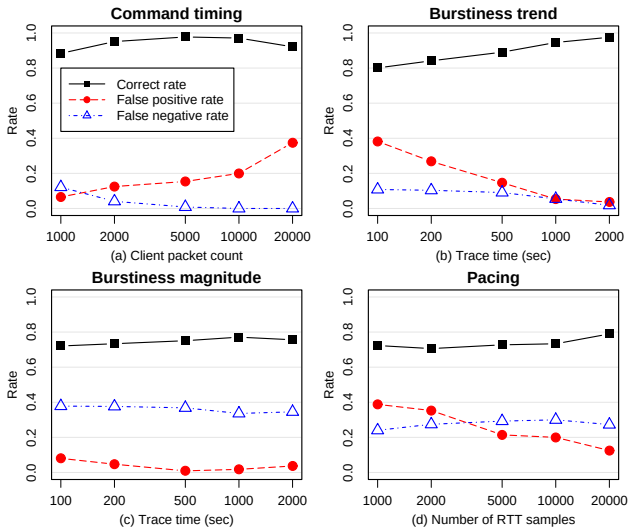

Figure 10: Evaluation results for the proposed decision schemes with different input size.

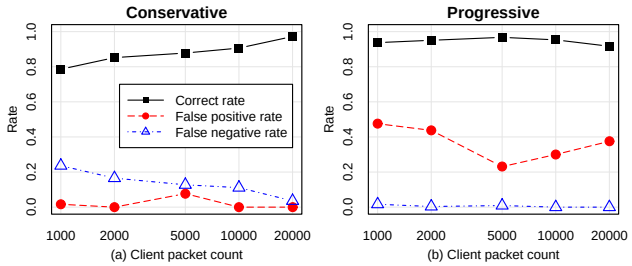

Figure 11: Evaluation results for the integrated schemes with different input size.

sion rate ($\approx 75\%$). Although the *pacing* method does not perform well, it is still proposed because of its unique relation to human behavior.

In practice, we can detect game bots using an integrated approach, i.e., by applying multiple schemes simultaneously and combining their results according to preference. For example, if a conservative judgement is preferred, a traffic stream could only be declared as corresponding to a game bot if all schemes agree with that declaration. By this reasoning, we propose two integrated schemes, a conservative approach and a progressive approach. Combining the *command timing* and *burstiness trend* methods, the conservative approach is achieved by a logical "AND" operation and the progressive approach by an "OR" operation. The performance of these integrated schemes, as presented in Fig. 11, is rather good in terms of reducing the occurrence of certain kinds of false alarm. The conservative approach reduces the false positive rate to zero and achieves a 90% correct decision rate, given an input size of $10,000$ packets. Meanwhile, the progressive approach produces a false negative rate of less than 1% and achieves a 95% correct decision rate, given an input size of $2,000$ packets.

8. DISCUSSION

In this section, we first discuss the generality of our proposed schemes, i.e, the possibility of applying them to other bots and games. We then analyze the robustness of the schemes under the presence of counter-strategies from bot developers.

8.1 Generality

As bots cannot actually "watch" screens, they perceive the environment by inspecting the information conveyed by server packets. Thus, the design naturally leads to that, whenever a bot receives a change of the game world, it will take some reactions immediately by sending back commands to the server. When a succession of commands, rather than a single command, needs to be sent, to not overwhelm the network and server, some pacing mechanism that can spread out the release time of commands will be used. Therefore, we argue that such design is not unique to the particular bots we studied, but commonly exists in MMORPG bots. In other words, the *command timing* scheme is generalizable as it is based on the fact that bots react to server packets with some form of regularity.

In terms of traffic burstiness, as bots take actions based on server packets, which are inevitably periodic for smooth screen updates, the periodicity will propagate into bot traffic. Therefore, we will certainly find lower burstiness in time scales around the state updating frequency. In addition, bots do not have human-like behavior such as heavy-tailed activities, so the large-scale burstiness of bot traffic will be lower than that of traffic generated by human players [3]. That explains why the patterns in multi-scale analysis of traffic burstiness (Section 5.2) should be generally observable, rather than a particular phenomenon in our settings.

The sensitivity to network conditions should be general since it reflects user reaction to the rate of screen changes, which is irrespective of the game design. On the other hand, a bot will not exhibit such human behavior as long as the release of certain commands is timer-based, where timers are usually unavoidable to schedule successive actions. Thus, unlike the case of human players, the command rate of bots will show no significant correlation with the pace of screen updates.

8.2 Robustness against Counter-Attacks

While the traffic analysis approach is particularly generalizable because it is independent of game design and content, the most challenges of traffic-based bot detection is the robustness of schemes under counter-attacks from bot developers. As our strategies use packet timestamps as the only input, an obvious counter-strategy is *adding random delays to the release time of client commands*. Random delays will undoubtedly make the *command timing* scheme ineffective, as this method relies on the regularity of bot reactions. However, we argue that the schemes based on traffic burstiness and human reaction to network conditions are robust under such attacks.

The reason that the *burstiness trend* scheme is immune to random-delay attacks because bots take actions according to state updates which are periodic in nature. Adding random delay to command timing will not eliminate the regularity unless the added delay is longer than the updating interval *by orders of magnitude or heavy-tailed*. However, adding such long delays will make the bots *incompetent* as

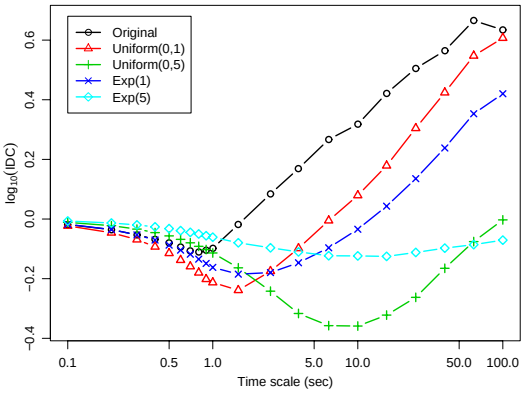

Figure 12: The IDC of the original packet arrival process in *Kore1* and that of intentionally-delayed versions

this will slowdown the character's speed by orders of magnitude. We illustrate the property via simulations. Using the trace *Kore1* as an example, we postpone the release of each command by random delays drawn from uniform and exponential distributions, respectively. The IDC of the original trace and that of intentionally-delayed versions are shown in Fig. 12. As can be seen, random delays do not remove the "dip" in the burstiness trend, it could only mitigate the extent of or change the location of the dip. On the other hand, the *burstiness magnitude* scheme examines the relative variability of client traffic versus that of server traffic. Traffic generated by human players are bursty because their high variability in activities [3]. It is certainly possible for a bot to mimic such human behavior by making its ON/OFF time close to heavy-tailed, but this will largely reduce the efficiency of the bot to reap rewards because of the non-significant probability of very long idle time. That is, even though bot developers can fool the *burstiness magnitude* scheme by incorporating heavy-tailed ON/OFF activities, the bots will bring much less threat to the balance of the game world (if they are not detected by other schemes).

Random delays have no effect on the *pacing* scheme because delays only increase the variance of the command rates, but not the average command rates. Bot developers may mimic human behavior by intentionally adapting the packet sending rate to the network conditions. There are two difficulties to implement this counter-measure in bots: 1) the pace of screen updates is decided by the transit delay of server packets, which is measurable for bot detection software run at the server side, but bots can only measure transit delay of client packets. 2) For TCP-based games, packet transit delay can only be obtained by intercepting packets from the TCP stack or installing a kernel-mode packet sniffer. The former obstacle makes a bot unable to exactly measure the pace of screen updating, even react to it. The latter brings much difficulty in development and deployment of bots as users often avoid installing kernel-level software unless obviously necessary.

9. CONCLUSION

In this paper, we have addressed the game bot problem in MMORPGs and proposed to identify game bots using traffic analysis. Taking *Ragnarok Online* as a case study, we traced and analyzed packet traces for mainstream game bots and human players with different network settings. We have shown that the traffic corresponding to bots and human players is distinguishable in various respects, such as the regularity and patterns in client response times, the trend and magnitude of traffic burstiness in multiple time scales, and the sensitivity to network conditions. We consider that the tendency of human players to unconsciously adapt to game pace is rather potential and worth further investigation.

Based on the findings in traffic analysis, we have proposed four decision strategies and two integrated schemes for the bot detection problem. For our collected traces, the conservative approach of our proposed integrated schemes reduces the false positive rate to zero and produces a 90% correct decision rate, given an input size of $10,000$ packets. The progressive approach, on the other hand, yields a false negative rate of less than 1% and achieves a 95% correct decision rate, given an input size of $2,000$ packets. We have also shown that the proposed methods are generalizable to other games and robust against counter-measures from bot developers.

Acknowledgments

The authors are indebted to Kiya Hsu, Gino Liu, and Chia-Chun Hung, who continuously played *Ragnarok Online* for more than five hours during the trace collection. The authors also wish to thank the anonymous referees for their constructive criticisms.

10. REFERENCES

[1] N. E. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. In *Proceedings of IEEE INFOCOM 2001*, Anchorage, AK, Apr. 2001.

[2] P. Bloomfield. *Fourier Analysis of Time Series: An Introduction*. John Wiley & Sons, New York, 2000.

[3] K.-T. Chen, P. Huang, and C.-L. Lei. Game traffic analysis: An MMORPG perspective. *Computer Networks*, 51(3), 2007. Article In Press.

[4] E. Cronin, B. Filstrup, and S. Jamin. Cheat-proofing dead reckoned multiplayer games. In *Proc. of 2nd International Conference on Application and Development of Computer Games*, Jan 2003.

[5] M. DeLap, B. Knutsson, H. Lu, O. Sokolsky, U. Sammapun, I. Lee, and C. Tsarouchis. Is runtime verification applicable to cheat detection? In *Proceedings of ACM SIGCOMM 2004 workshops on NetGames '04*, pages 134–138. ACM Press, 2004.

[6] R. A. Fisher. Tests of significance in harmonic analysis. *J. Roy. Stat. Soc., Ser. A*, 125:54–49, 1929.

[7] W. A. Fuller. *Introduction to statistical time series*. John Wiley & Sons, 1996.

[8] R. Gusella. Characterizing the variability of arrival processes with indexes of dispersion. *IEEE J. Select. Areas Commun.*, 9(2), Feb. 1991.

[9] J. Hartigan and P. Hartigan. The dip test of unimodality. *Ann. Stat.*, 13:70–84, 1985.

[10] P. M. Hartigan. Computation of the dip statistic to test for unimodality. *Appl. Stat.*, 34(3):320–325, Sept. 1985.

[11] M. Kendall. A new measure of rank correlation. *Biometrika*, 30:81–93, 1938.

[12] B. S. Woodcock. An analysis of MMOG subscription growth – version 18.0.

[13] J. Yan and B. Randell. A systematic classification of cheating in online games. In *Proceedings of ACM SIGCOMM 2005 workshops on NetGames '05*. ACM Press, 2005.