Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 797159, 22 pages
doi:10.1155/2009/797159

Research Article

Identifying MMORPG Bots: A Traffic Analysis Approach

Kuan-Ta Chen,[1] Jhih-Wei Jiang,[2] Polly Huang,[3] Hao-Hua Chu,[2] Chin-Laung Lei,[3] and Wen-Chin Chen[2]

[1] *Institute of Information Science, Academia Sinica, Taipei 115, Taiwan*
[2] *Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan*
[3] *Department of Electrical Engineering, National Taiwan University, Taipei 106, Taiwan*

Correspondence should be addressed to Kuan-Ta Chen, ktchen@iis.sinica.edu.tw

Received 10 April 2008; Accepted 8 September 2008

Recommended by Rocky Chang

Massively multiplayer online role playing games (MMORPGs) have become extremely popular among network gamers. Despite their success, one of MMORPG's greatest challenges is the increasing use of game bots, that is, autoplaying 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 versus game bots and propose general solutions to identify game bots. Taking *Ragnarok Online* as our subject, we study the traffic generated by human players and game bots. 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 different network conditions. Based on these findings, we propose four strategies and two ensemble schemes to identify bots. Finally, we discuss the robustness of the proposed methods against countermeasures of bot developers, and consider a number of possible ways to manage the increasingly serious bot problem.

1. Introduction

Massive multiplayer online role playing games (MMORPGs) 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 doubled between July 2004 and January 2006 to a 13-million player base [1]. Despite their success, one of MMORPG's greatest challenges is how to maintain the subscription base in the face of the increasing use of game bots (http://en.wikipedia.org/wiki/MMORPG#Bots).

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 does not necessarily exploit any bugs or vulnerabilities of the game software; it just "plays" the game in place of a human. Currently, bots must identified *manually* by launching a dialogue with a suspect character, as a bot cannot speak like a human. However, this method leads to a significant administrative burden. In this paper, we analyze the traffic generated by human players versus game bots and propose general solutions to identify game bots automatically. To the best of our knowledge, this is the first work to investigate automatic, game-independent, bot identification techniques by using network traffic analysis.

Taking *Ragnarok Online* (Ragnarok Online, http://iro.ragnarokonline.com/), one of the most popular MMORPGs in the world, as a case study, we analyze the traffic of human players and mainstream game bots under different network settings. We find that traffic generated by bots versus 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 ensemble 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 achieves 95% accuracy. The former completely avoids making false accusations against bona fide players, while the latter tracks game bots down more aggressively. We show that the proposed methods are generalizable to other bots and games, and that they are robust against simple random-delay counter-measures of bot developers.

However, we also showed that the pure traffic-based detection schemes might be deceived by sophisticated counter-attacks that mimic human gaming activities. This situation cannot be avoided completely because game bots can always imitate human activities at the network level. Given the intrinsic difficulties of automatic bot identification, we believe that the most effective bot detection scheme should be multimodal rather than a single approach. To this end, we explore a number of promising strategies that work at higher semantic levels.

The remainder of this paper is organized as follows. Section 2 reviews related works. Section 3 provides a brief introduction of the game *Ragnarok Online* and an assessment of the current status of game bots. Section 4 discusses the trace collection. Section 5 characterizes 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. In Section 9, we explore a number of potential strategies for managing bots at higher levels (in contrast to the network level). Section 10 contains 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 schemes [2–5]. Since game cheats often exploit loopholes in game rules or specific implementations, researchers attempt to guarantee the integrity of game systems by, for example, runtime verification of transaction atomicity [4]. However, the proof of correctness approach is not applicable to bot detection problems because game bots do not necessarily "cheat." Some game bots cheat by exploiting bugs or reading process memory, while others do not. Noncheating game bots work just like regular players; they cannot do anything regular players cannot do. The difference between a bot-controlled character and a human-controlled character might only lie in the qualities of humanness and intelligence that the former lacks.

A number of studies have employed machine learning techniques to detect game bots in online games. For example, Yeung et al. [6] proposed using the a *dynamic Bayesian network* (DBN) to model the aiming accuracy for *aimbot* detection in first-person shooter (FPS) games. In the DBN, the aiming accuracy depends on whether the player is cheating, whether the player or the target is moving, the aiming direction, and the distance between the player and the target. The model can detect cheaters with a high degree of accuracy, but it can only be applied to aimbots. Kim et al. proposed detecting *auto programs* in MMORPGs [7] based on the window events, which are generated by a player's key strokes, mouse button clicks, and mouse movements. Various classification schemes, such as the decision tree, the *k*-NN classifier, the multilayer perceptron network, and the naive Bayesian classifier, are used to determine whether automated programs are being used. Because of the high regularity exhibited by such programs, the window-event-based approach yields a decent performance irrespective of the classification method used. Chen et al. [8, 9] proposed using avatars' movement trajectories to detect the use of game bots in FPS games. Their rationale is that the trajectory of an avatar controlled by a human player is hard to simulate. Since human decisions on avatar movements may not always be logical and efficient, how to model and simulate realistic movements is still an open question in the AI field. Chen et al. show that game bot detection based on the spatial and temporal characteristics of the avatars' trajectories is effective in Quake 2.

Golle and Ducheneaut proposed using completely automated public turing test to tell computers and humans apart (CAPTCHA) tests [10], either software-based or hardware-based, to prevent bots from playing online games [11]. Currently, the most widely used CAPTCHA tests currently rely on the ability of human beings to recognize randomly distorted text or images. The drawback of this approach is that the cost of bot detection must be distributed among all users, including legitimate players, as the tests will inevitably interrupt players' adventures and reduce their sense of immersion in the virtual world. Legitimate players do not normally like these kinds of tests as they may feel they are suspected of cheating. This could be one reason that password-book-based anticopy mechanisms for PC games are no longer popular. However, these mechanisms might be the best candidates for the last line of defense if automatic bot identification mechanisms, such as our proposed traffic-based schemes, are used. In other words, passive detection schemes could be used initially to find suspects among the thousands of honest players, after which CAPTCHA tests could be applied to the suspected characters.

Recently, anticheating softwares, such as PunkBuster and GameGuard, have been widely deployed in online games to prevent cheating. Such software is bundled with game clients, and cannot be uninstalled even if the game client is uninstalled. It works by hiding the game client process, monitoring the entire virtual memory space (to prevent modification of the game executable image), blocking suspected programs that might be hacker tools, and blocking certain API calls. This kind of software can neutralize nearly every plug-in tool that attempts to hook the game client program in order to inspect or modify game states when the game is running; however, it cannot stop the widespread of standalone bots, including the bot series we study in this paper. The reason is obvious; anticheating softwares are host-based, so they must be installed on players' PCs to be effective. In contrast, standalone bots can be executed *without* game clients; therefore, anticheating tools cannot prevent game bots as they would not normally be installed on PCs where standalone bots are running. This is evidenced by the fact that game bots may still be active in games protected by PunkBuster or GameGuard, for example, Quake (PunkBuster) and Lineage (http://boards.lineage2.com/showflat.php?Number=573737.) (GameGuard).

3. Ragnarok Online and the Bots

The core features of most MMORPGs are more or less standard, for example, 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 operations, 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; then, the masters can decide if the suspicious characters are actually bot-controlled or human-controlled based on their responses. However, given millions of online players, this method is very inefficient and incurs a significant administrative burden. The biggest drawback is that the detection is *intrusive*, so it may offend innocent players. As the problem of players cheating with game bots becomes more rampant and serious, we believe the demand for automatic bot identification techniques for online games is urgent.

We surveyed publicly-available game bots for *Ragnarok Online*, and found that although more than a dozen bots are available, most of them are derived from the well-known *Kore* project (Kore, http://sourceforge.net/projects/kore/). *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, for example, *KoreC*, *X-Kore*, *mod-Kore*, *Solos Kore*, *wasu*, *Erok*, *iKore*, and *VisualKore*, have been developed from it. Se also found that a similar bot program, DreamRO (DreamRO, http://www.game186.com/SoftList/Catalog_76_SoftTime_Desc_1.html) and its derivatives, is very popular in China and Taiwan.

Both *Kore* and *DreamRO* are standalone bots, that is, 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. In addition, they both allow users to give commands anytime, regardless of the prearranged actions of the scripts, that is, the bots are both script-based and interactive.

4. Trace Collection

To develop bot identification techniques based on 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 the trace collection tractable, we chose a bot program to represent each series. *KoreC*, the Chinese edition of *Kore*, was selected to represent the *Kore* series, and *DreamRO* was chosen to represent the *DreamRO* series.

We collected a total of 19 game traces at the client side, that is, the traffic monitor was attached to the same LAN as the game clients. 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 and slow access links, and the network media ranged from Fast Ethernet to ADSL. 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, resp.) without advanced skills or powerful weapons. The scripts we used for the two bots are commonly available in 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 killing a monster, the bot will take the loot, and turn to another monster. The process will continue until the backpack is full of loot. At that time, the bot will go 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 proficiency levels and professions.

In total, the collected game traces (The complete game traces (in `tcpdump` format) are publicly available at http://mmnet.iis.sinica.edu.tw/content.html?key=ro.) contain 3 million packets over 206 hours, as summarized in Table 1. For brevity, we denote the four players as A, B, C, and D, respectively, Kore as K, and DreamRO as R. Traces from the same bot/player are coded by a unique digit following the bot/player's identifier. The period of a game trace indicates the continuous gaming time. We asked the human players not intentionally leave their characters idle during the game

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 2 M/512 Kbps	1.8 hr	12	51,823	3.5 MB	0.9 / 3.9 pkt/s	82.0 ms	0.03%
		A2		5.6 hr	14	147,814	10.5 MB	0.8 / 3.4 pkt/s	95.4 ms	0.03%
	Kiya	*B1*	APOL 2 M/512 Kbps	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	ETWEBS-TW	32.7 hr	37	271,416	13.3 MB	0.6 / 0.7 pkt/s	96.5 ms	0.004%
		K4		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	ETWEBS-TW	42.3 hr	42	652,877	34.1 MB	0.8 / 1.8 pkt/s	85.3 ms	0.05%
		R4		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 (2 M/512 Kbps), ETWEBS-TW (2 M/256 Kbps), and TANET (100 Mbps).
‡ Packet rate column format is "client data packet rate/server data packet rate," that is, pure TCP ack packets do not count.

as client traffic will not be generated if game characters are left idle. Each game trace is composed of a number of TCP connections, where each connection corresponds to the activity within the same map. The game world of *Ragnarok Online* is partitioned into a number of maps, provided by several map servers. When a character moves across map boundaries (by walking, transport, or teleporting), the game client will disconnect from the original map server and establish a connection with the new map server. Therefore, the number of connections implies the number of map switches, which indicates how frequently a character moves across maps. The packet rate column lists the average rate that data packets are sent by game clients and servers. The average client packet rate indicates the type of *player activity*, since each player command is conveyed by a client data packet. On the other hand, the average server packet rate indicates the level of *interaction*, that is, the popularity of and the amount of activity in the area where the character resides, as server packets convey information about the activities of characters nearby [12]. Note that the average packet rate is roughly the same for the same bot/player under the same network setting, which may be seen as a "signature" of the game playing behavior of a certain bot/player. Based on these two metrics, we show that the behavior of our selected human players is heterogeneous. In addition, the average round trip times (RTTs) and packet loss rate statistics manifest the heterogeneity of the network conditions experienced during the traced sessions.

5. Characterization 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 commands, for example, that approach another character, attack a nearby monster, or cast healing magic, are triggered by keyboard strokes or mouse clicks. In contrast, for game bots, triggering client commands are decided by the decision engine in the bot program. Thus, a bot's decision about *when to issue the next command* is critical to us because it leads to major discrepancies in traffic patterns between different bot series, as well as between bots and human players.

Our analysis of the release timing of client commands from bots shows that the release of commands, for both *Kore* and *DreamRO*, relates to 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*, for example, continuous slashing until an enemy is defeated. A series of successive commands are also usually implemented with timers, for example, when the life point is lower than a certain threshold, a character must immediately drink a healing potion, and then cast protective magic at himself and destructive magic at the most threatening enemy. Using a timer to schedule the above commands sequentially with certain intervals is the most common design. On the other hand, since server data packets carry the latest status about

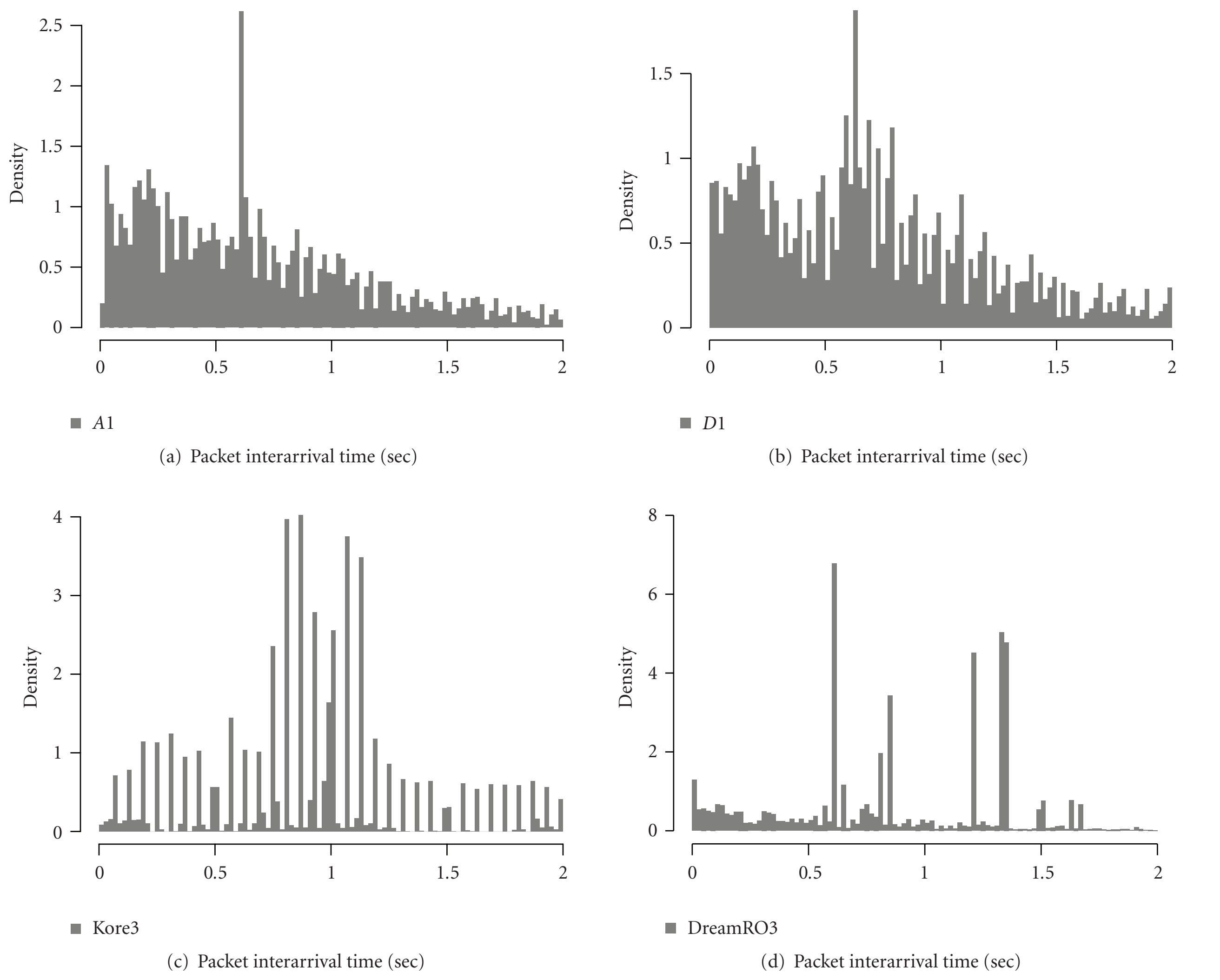


FIGURE 1: Histogram of packet interarrival times.

the character and environment, for example, 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 would issue movement commands continuously whenever it learns 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 arrival processes. Lastly, we identify the particular patterns in human behavior caused by sensitivity to network conditions, which, of course, game bots do not possess.

5.1. Regularity in Client Traffic. Figure 1 shows the histograms of client packet interarrival times shorter than 2 seconds. While player traces in the upper two plots show randomness in packet interarrival times, the bot traces, shown in the lower two plots, suggest the existence of a timer triggering mechanism and the absence of randomness that characterizes human actions. Specifically, *Kore3* reveals a periodic timer of 16 Hz, that is, most of the packet interarrival times are multiples of 1/16 second, while *DreamRO3* displays more regular timing, as most of the interpacket times concentrate on certain values.

An empirical cumulative distribution function (CDF) plot of packet interarrival times manifests the above statements more clearly. In Figure 2, the CDF curves of player traces, *A1* and *B2*, increase smoothly, except for a sudden rise around 0.6 seconds, which is a frequency component inherent in game clients. We also provide the CDF curve of an exponential random variable fitted to *A1* by maximum likelihood estimation (MLE). Though the exponential curve does not fit the empirical CDF of *A1* very closely, it can be seen an approximation. On the other hand, the curves of *Kore1* and *DreamRO3* show *zigzag* patterns, which strongly suggests that packet interarrivals in both bot traces are concentrated around certain times.

5.1.1. Entropy of Packet Interarrival Times. From the above observation, we find that the distribution of packet interarrival times is more regular for bot traffic than player traffic. We now attempt to exploit this property to distinguish bot traces from player traces.

A well-known metric for judging the degree of randomness of a variable is its degree of entropy. We start with the definition of Shannon's entropy, a traditional measure of the uncertainty in a random variable. The Shannon's entropy, $H(x)$, of a discrete random variable, x, that takes on the value v_i with probability p_i is defined as

$$H(x) = -\sum_i p_i \log_2 p_i. \quad (1)$$

We compute the entropy of each trace by segments, that is, the interpacket times of each trace are first divided into several segments, and the entropy is computed for each segment separately. The computed entropy with segment size 5000 for all traces is depicted in Figure 3. The result conforms to our observation that the entropy of player traces is mostly higher than that of bot traces. We provide a possible threshold on the plot so that the entropy of player traces is higher than the threshold, while the entropy of bot traces is lower than that. However, choosing an appropriate threshold is difficult, because we cannot decide how large is "large" for the computed entropy. Furthermore, if we compute the entropy with larger segments, the entropy between bot and player traces will be less distinguishable. This is because with more packets, there is more chance of randomness in the packet interarrival times of bot traces. For these reasons, we do not use the entropy of packet interarrival times to distinguish between bots and human players.

5.1.2. Frequency Components. By incorporating the time factor, we take the successive packet interarrival times in a trace as a time series. We find that in some bot traces, packet interarrival times occur periodically in a statistical sense. For example, Figure 4(a) depicts 100 successive interpacket times in the trace *DreamRO3*. On the graph, there is a pronounced pattern, wherein one or two large packet gaps of approximately 2.5 seconds occur about every 10 packets. Such a regular pattern is a consequence of bot behavior for certain tasks; for example, in its monster-hunting process, a bot will move in a randomly chosen direction for certain steps, and repeat the steps until a monster is within its view scope. This time series can be viewed in the frequency domain by a transform to the corresponding power spectral density function, as shown in Figure 4(b). On the graph, frequency components of 0.1 Hz and 0.2 Hz are clearly present, where the 0.1 Hz frequency corresponds to the 10-packet-period in Figure 4(a). Note this phenomenon appears in both the *Kore* and *DreamRO* traces, but it is not present in player traces. However, not all bot traces exhibit pronounced frequency components, since the proportion of regular behavior is small compared to the whole trace. Therefore, we

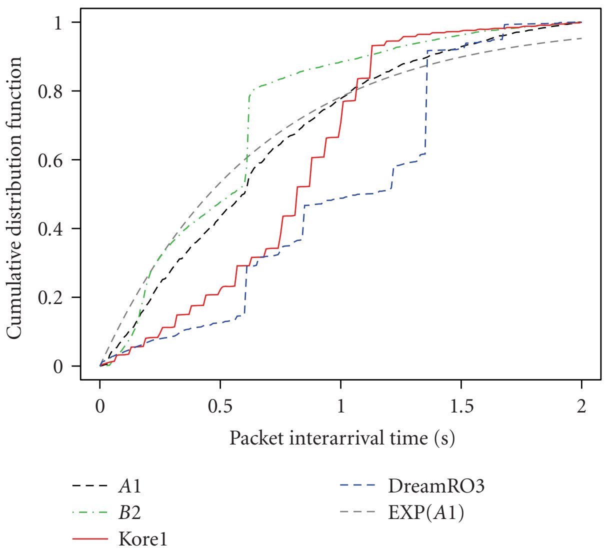


FIGURE 2: CDF of packet interarrival times.

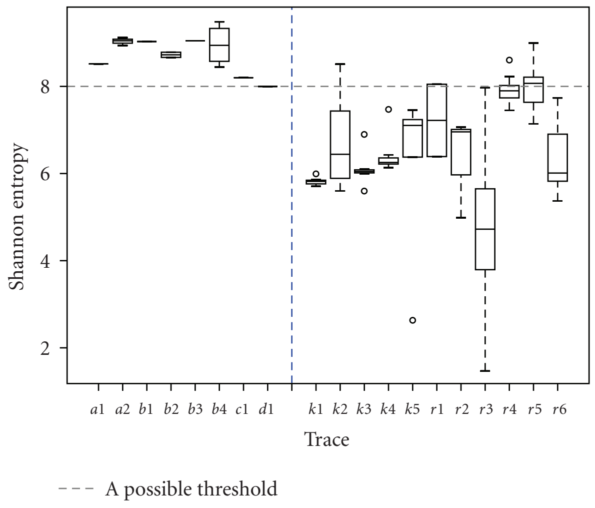


FIGURE 3: Entropy computed from packet interarrival times for each trace. Each group of 5000 interpacket times is computed separately. The threshold is chosen arbitrarily to reveal that the entropy of player traffic is almost always higher than that of bot traffic.

do not consider that periodicity in packet interarrival times is an effective method for recognizing game bots.

We can use another metric to check the periodicity in traffic, that is, the frequencies embedded in packet arrival processes. For each trace, we obtain the corresponding packet arrival process by counting the number of client packets released every 0.1 seconds. Since for each trace, at every instant exactly one connection is active, we argue that the corresponding packet arrival process is just stationary, regardless of the rate variation during game playing. A preliminary check of player traces shows that at least three strong frequencies are inherent in the game design, namely, 1/12 Hz, 1.67 Hz, and 60 Hz. This behavior echoes the study of another MMORPG [12], *ShenZhou Online*, which shows

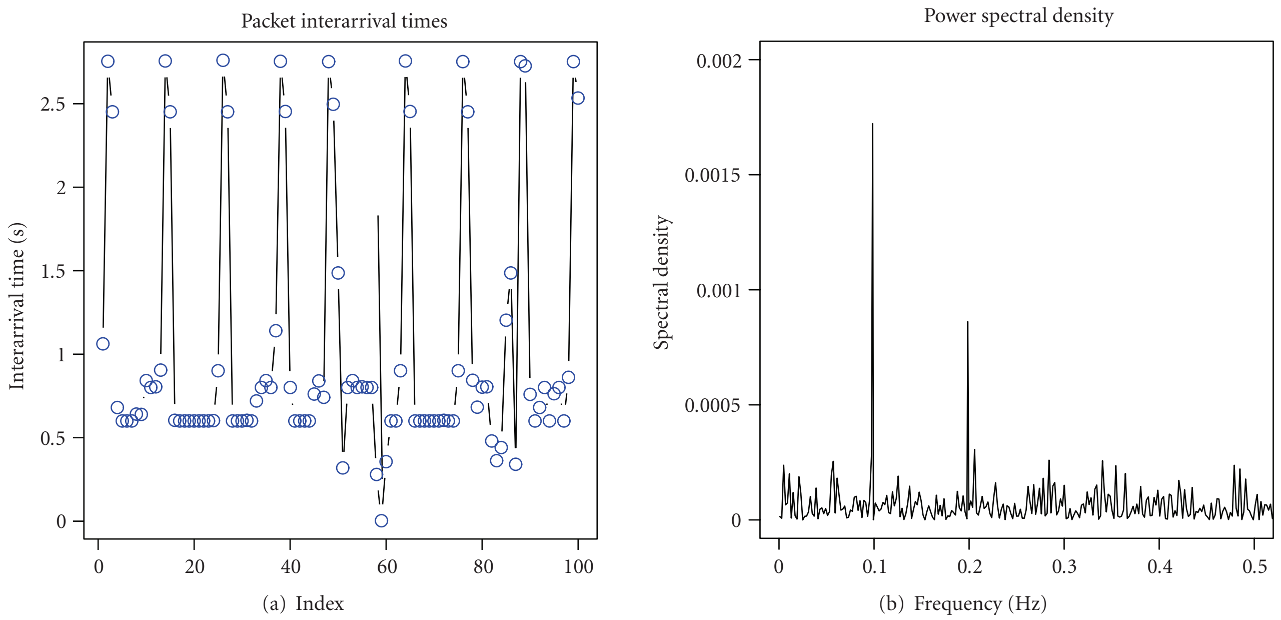


FIGURE 4: (a) *DreamRO3* exhibits regular packet interarrival times, such that, on average, one or two large packet gaps occur for every 10 packets. (b) Power spectral density of packet interarrival times.

that significant frequency components exist in game traffic in either direction. Comparing the power spectrum of bot traces with that of player traces, we find that bots induce additional frequency components not present in player traces, as shown in Figure 5. However, since the frequencies in game traffic may be adjusted based on a character's attributes [12], for example, race, skill, or equipment, and since we do not have complete knowledge of all possible frequencies built into the design of *Ragnarok Online*, we cannot decide whether a frequency component is *inherent* in the game design or *induced* by a bot program. For this reason, we do not simply use the frequency components of packet arrival processes to identify game bots.

5.2. Command Timing. We begin by defining the "client response time" as the *time difference between a client packet's departure time and the most recent server packet's 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, we plot the cumulative distribution functions of response times of less than 0.1 seconds for four traces, as shown in Figure 6. In the figure, except for an initial rise for *A1*, the two player traces, *A1* and *B2*, are similar in that their response times of less than 0.1 seconds increase smoothly, that is, they are almost uniformly distributed. On the other hand, bot traces reveal different patterns; the CDF of *Kore1* is a zigzag-type, that is, 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, that is, strong modes and zigzag CDF, in more depth.

5.2.1. Quick Response. Among all game traces, only those of *DreamRO* possess a considerable number of short response times, for example, $\leq$ 10 milliseconds, which we call *quick responses*. These responses are frequent enough and clustered so that more than one peak is formed in the corresponding histogram, as shown in Figure 7. Note that to distinguish peaks clearly we take a logarithm of the response time. The quick response manifests that *DreamRO* often issues client commands immediately upon the receipt of server packets, while *Kore* employs a more sophisticated command timing mechanism.

5.2.2. Regularity in Response Times. Although quick 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 Figure 8, which depicts histograms of response times shorter than 0.5 seconds, 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 the 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 16 milliseconds for both *Kore* and *DreamRO* (equivalent to 60 Hz). In Section 6.1, we will propose a bot detection scheme based on the quick responses and the regularity in response times identified above.

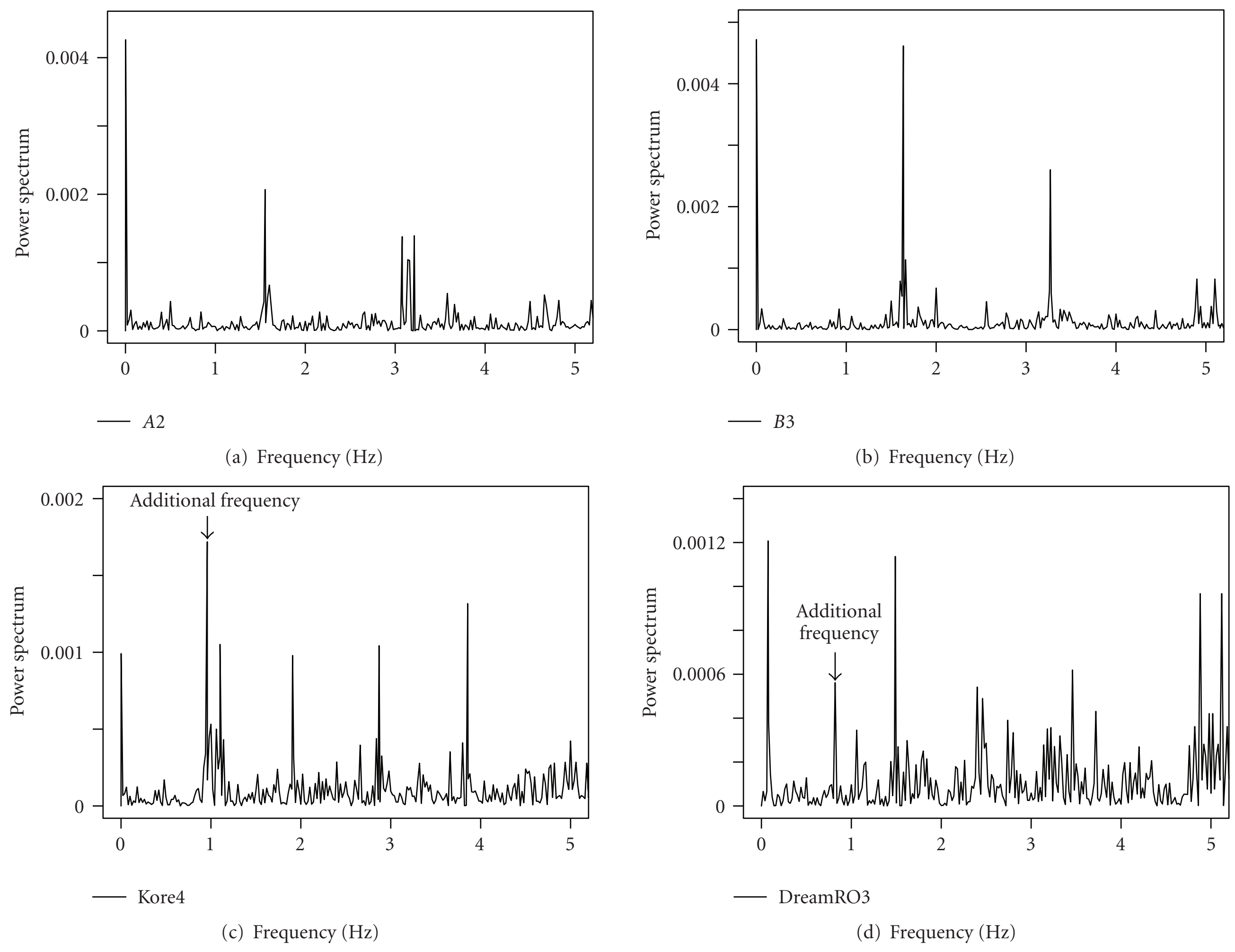


Figure 5: Power spectral density of packet arrival processes. Frequencies embedded in player traces are alike; however, bot traces display additional frequency components that do not appear in player traces.

5.3. Traffic Burstiness. Traffic burstiness, that is, the variability of byte or packet counts 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* (or smooth) the bot traffic is. Our hypothesis is that a bot, by virtue of its periodicity, should exhibit smoother traffic compared with those of players. In the following, we use the index of dispersion for counts (IDCs) to quantify the variability of traffic over different time scales.

There are several commonly used metrics of traffic burstiness [13]. In the following, we first evaluate the coefficient of variation (CoV) of packet interarrival times, and then use the index of dispersion for counts (IDCs) to capture the variability of traffic over different time scales.

5.3.1. Coefficient of Variation. The coefficient of variation (CoV) of packet interarrival times is defined as the ratio of the standard deviation of the interarrival times to the expected value of interarrival times. The CoVs of selected game traces, computed with a segment size of 500, are plotted in Figure 9. By definition, the CoV of any exponential random variable is equal to 1, as its standard deviation is always equal to its mean. From the graph, the average CoVs of player traces are all higher than 1, while most bot traces have CoVs lower than 1. However, we do not consider that the CoV is an effective indicator for differentiating bots and players because of randomness. For larger segments, all game traces tend to have CoVs higher than 1, so the boundary between bot traces and player traces is difficult to determine. Thus, in the following, we use a more sophisticated method to measure traffic burstiness by characterizing it in various time scales.

5.3.2. Index of Dispersion for Counts. Like all other software programs, a bot program must have a *main loop*, where each iteration of the loop corresponds to a minimum unit of operation, for example, issuing a command for a character, or processing a server packet. The rationale behind multitime-scale burstiness analysis is that, assuming each iteration (of the main loop) takes approximately the same amount of time, and the game bot sends out roughly

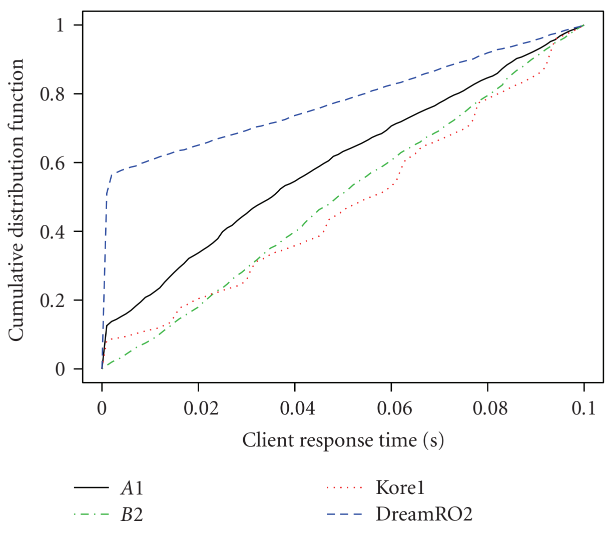


Figure 6: CDF of client response times.

the same number of packets in each iteration, then *traffic burstiness will be lowest at the time scale equal to the amount of time needed for each iteration of the main loop*.

We use the index of dispersion for counts (IDCs) to measure traffic burstiness in multiple time scales. 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 [14], that is,

$$I_t = \frac{\mathrm{Var}(N_t)}{E(N_t)}, \qquad (2)$$

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

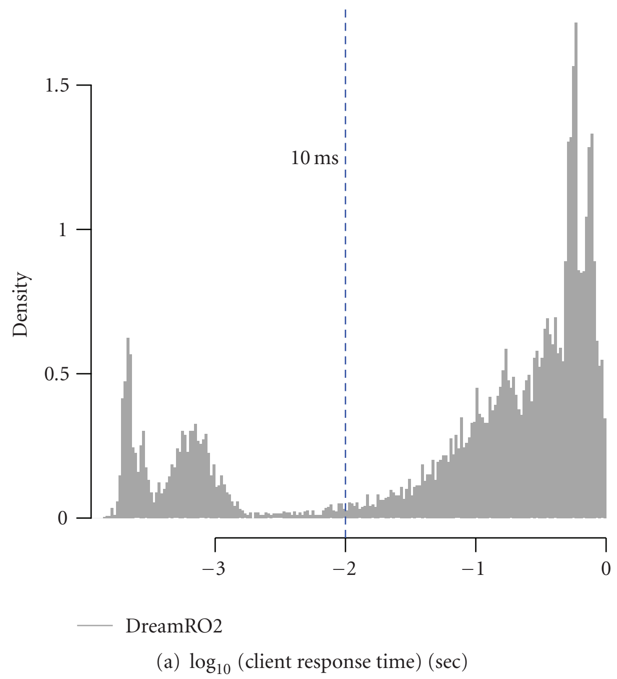
The IDCs for selected game traces, with the Poisson rate regulation heuristic applied, are plotted in Figure 10. We make two observations from the plots (1) bot traffic is smoother than player traffic, but it is hard to define a threshold for the burstiness magnitude, and (2) all bot traces support our 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; however, after a certain time scale with the lowest burstiness, a "rising trend" will appear. In contrast, the burstiness trends of most player traces increase monotonically in time scales >1 second. We 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 very different, game servers still treat all clients *equally*, that is, 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 average 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. Figure 11 shows the burstiness comparison for selected traces; 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. We exploit this property further to identify bots in Section 6.3.

5.4. Sensitivity to Network Conditions. The last aspect we consider is the *subconscious human reactions to network conditions* embedded in traffic traces. This is considerably different to previous approaches. We find that *human players adapt to the game pace involuntarily*. While a game client relies on server packets, which convey the latest information about other characters and the environment, to render its screen, its update speed is inevitably affected by network conditions. In short, we conjecture that a user's playing pace will be affected by the game update rate, which in turn is influenced by the transit delay of server packets. To evaluate how network delay affects a player's pace, samples of round trip times (RTTs) as well as the average packet rate in the next second following each RTT sample are computed. The plots describing the relationship between average packet rates and RTTs, where the latter are grouped in units of 10 ms, are depicted in Figure 12.

First, we analyze the player traces shown in Figures 12(a) and 12(b). The trend is clearly *downward*. This indicates that human players unconsciously slow down their keyboard and mouse actions to adapt to the slower game paces which are caused by severely delayed server packets. Figures 12(c) and 12(d) show that the same phenomenon does not occur in bot traffic; both the *Kore* and *DreamRO* traces show an *upward* trend in the relationship between the packet rate and RTT. Since bots have their own pacing schemes (certain frequencies dictated by timers), their pace is not *affected* by server packets like those belonging to 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 dashed vertical line on the graph denotes the median of the RTT samples. We find that our traces conform to the above observations for RTT samples smaller than the median RTT. One explanation is that higher RTTs are spread more diversely so that the number of samples in each group is not large enough to provide a robust statistic. Another possibility is that higher RTT samples are related to possible packet loss and retransmission, which could shrink the size of the TCP congestion window size. This in turn regulates the maximum packet rate. For these reasons, we restrict our analysis to RTT samples lower than the median. The pacing property of human players will be further exploited in Section 6.4 as a means of distinguishing bots from human players.



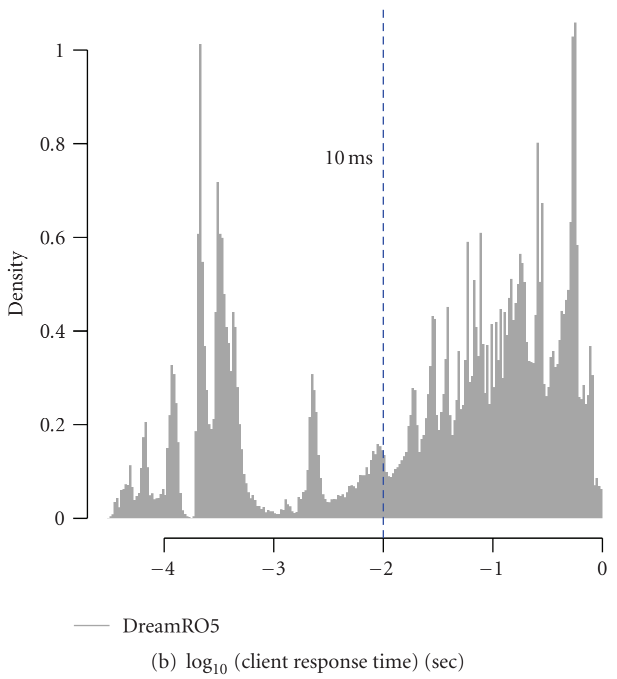


Figure 7: Histograms of client response times in two *DreamRO* traces. More than one peak is formed at time scales smaller than 10 milliseconds. Peaks do not occur in player traces or *Kore* traces.

6. Proposed Bot Detection Strategies

Here, 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 strategies is provided in the next section.

6.1. Command Timing. From the traffic characterization in Section 5.2, we find that client response times (following the receipt of server packets) from game bots are either extremely short because bots react to server packets immediately, or regularly spaced out because timers are used. Our first method, *command timing*, is based on these properties of the client response time. In this scheme, we simultaneously apply two tests (1) whether *multiple peaks* exist in the histogram of client response times that are less than 10 milliseconds, and (2) whether *regularity* exists in response times that are less than one second. The scheme returns true if either test is true, and false otherwise.

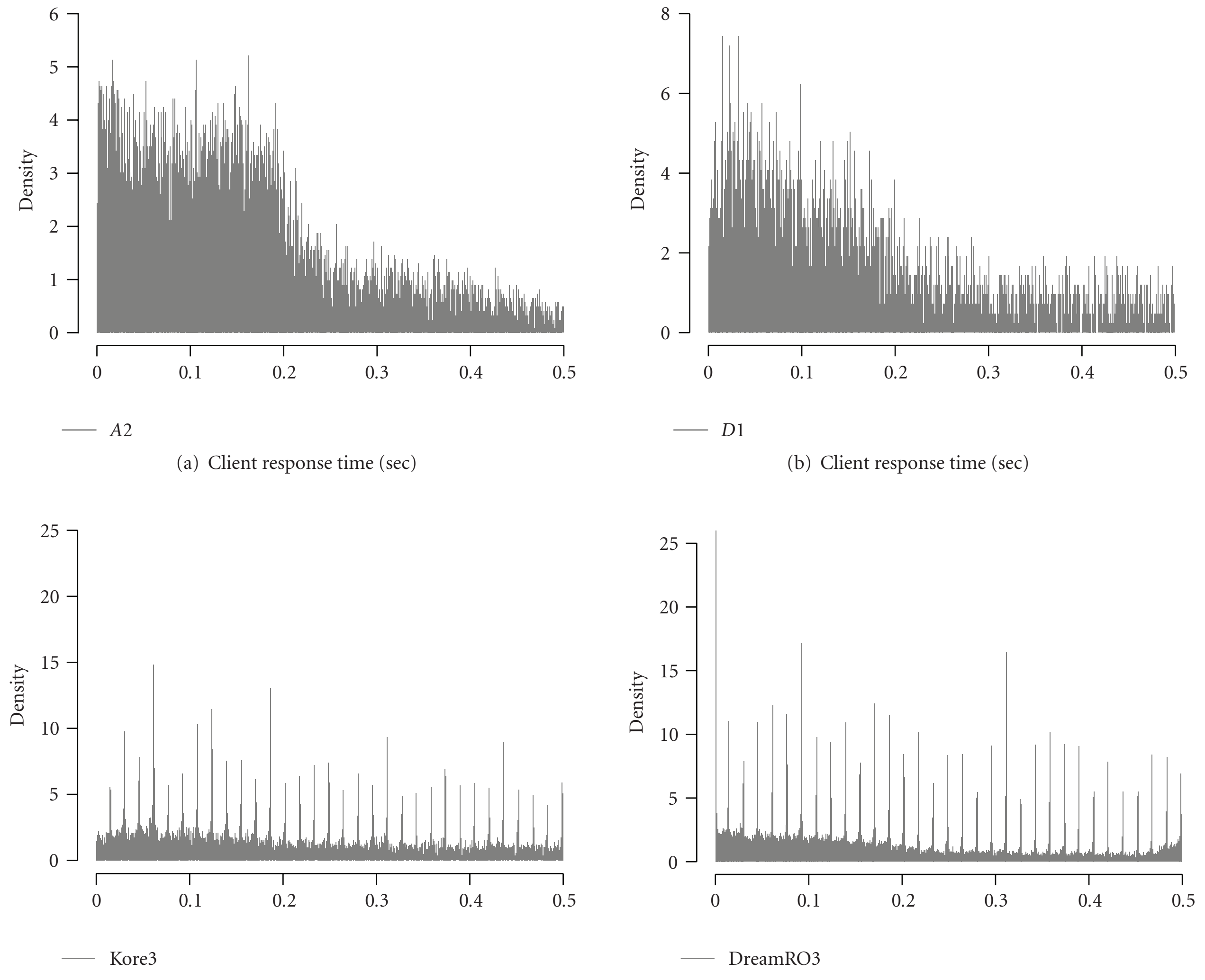
6.1.1. Multimodality Test. To detect the multimodality of response times less than 10 milliseconds, we use the Dip test [15, 16], which is designed to test unimodality. We first identify all local peaks and troughs in the response time histogram; then, for each candidate mode, which is determined by two troughs with at least one peak in between, we apply the unimodality test to the candidate, that is, to determine if the Dip statistic is significant at the 0.05 level. The multimodality test is deemed successful if and only if we can identify two or more modes with response times smaller than 10 milliseconds. Using this test with a segment size of 10000, we can correctly distinguish *DreamRO* bots from all other client types in most cases, as shown in Figure 14(a).

6.1.2. Regularity Test. As discussed in Section 5.2.2, client response times in bot traces show highly regular patterns in the form of response times clustered in multiples of a certain value (cf. Figure 8). To verify 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

$$I(f) = n\left|d(f)\right|^2, \tag{3}$$

where $d(f)$ is the discrete Fourier transform of that series at frequency f, and $I(f)$ is known as the periodogram. In order to exclude client packets that were not issued in response to the arrival of server packets, only response times shorter than one second are considered. Figure 13 shows the corresponding periodograms of the histograms in Figure 8. The strong spikes in the periodograms for bot traces are clear evidence of regularity in the response times.

We adopt the Fisher test to judge whether periodicity exists in periodograms, which is equivalent to the existence of regularity in the response times. Fisher [17] 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 of the Fourier frequencies



(a) Client response time (sec)

(b) Client response time (sec)

(c) Client response time (sec)

(d) Client response time (sec)

Figure 8: Histogram of client response times shorter than 0.5 seconds.

to the sum of the ordinates. Fuller [18] proposed an equivalent statistic, namely, the ratio of the largest ordinate to the average of the ordinates. While the null hypothesis is that the data consists of white noise, [19, Section 6.8] suggests 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, $I_1, I_2, \ldots, I_m$ will be independent and exponentially distributed with mean σ^2; that is,

$$F\left(\frac{I_j}{\sigma^2}\right) = 1 - e^{-x}, \quad x \geq 0, \; j = 1, 2, \ldots, m, \tag{4}$$

where $F(x)$ denotes the cumulative distribution function of x.

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

$$\Pr(\xi_m \leq \xi) \approx \exp(-me^{-\xi}). \tag{5}$$

Using this method, we test the regularity in response times. We consider that a trace corresponds to a bot if Fuller's statistic is significant at the 0.01 level.

The identification outcome, computed with a segment size of 2000 (as shown on the box-and-whisker plot in Figure 14(b)) indicates that the result is correct in most cases, although there are some misjudged cases.

6.2. Trend of Traffic Burstiness. We now turn to the second identification strategy. In this scheme, 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 5.3).

To check whether the burstiness initially exhibits a falling trend followed by a rising trend, we use the Mann-Kendall correlation test [20] 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

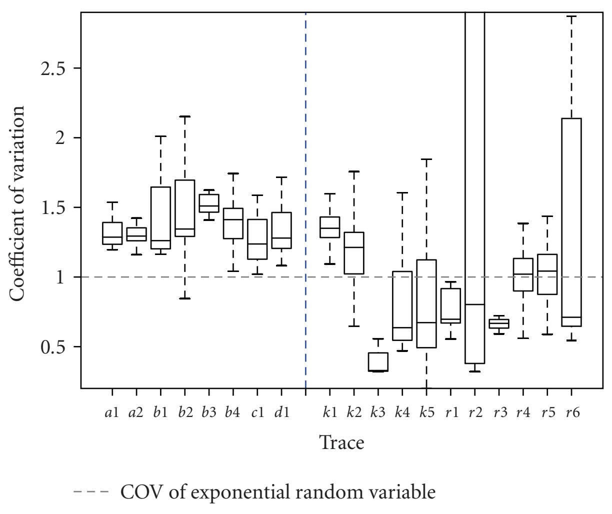


Figure 9: Coefficient of variation computed from packet interarrival times for each trace. Each group of 500 interpacket times is computed separately.

of variables, not on their values directly. Given the IDC ordinates, $\{I_t\}$, where $t > 0.1$ is the corresponding time scale, this scheme comprises two subtests. (1) Whether (t, I_t) exhibits a significant falling trend followed by a significant rising trend (both at a significance level of 0.05), and whether both trends can be detected in time scales smaller than 10 seconds. (2) Whether any time scale $t' > 10$ exists such 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. The results demonstrate that, except for a few outliers, the decisions of this scheme are mostly correct, as shown in Figure 15.

6.3. Magnitude of Traffic Burstiness. As described in Section 5.3 and exemplified in Figure 11, the burstiness of client traffic is relatively smoother for bots, compared to that of the corresponding server traffic. Moreover, 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 implemented as follows. For a given packet trace, we compute the IDCs for the client and server traffic, and search for the cross-point. If there is no cross-point, that is, the client traffic is always more bursty than the server traffic, we set the cross-point to the maximum time scale we use, which is 100 seconds. In Figure 16, we plot the cross-points for all game traces using a segment size of 10000. 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; otherwise, it corresponds to a human player. In most cases, the decisions are exact for player traces; however, some bot traces are classified as human players, especially for the *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 activities of characters nearby. Nevertheless, this method merits our attention as it yields the minimum false positive rate, that is, the number of times of a player is mistaken for a bot.

6.4. Reaction to Network Conditions. In Section 5.4, we investigated the relationship between round trip times and the corresponding packet rate; that is, *human players subconsciously adapt to network delay*; 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. Accordingly, we now propose our final scheme. For a given trace, we first take the RTT samples and the corresponding packet rates in the next second of the occurrence time of RTT samples. Then, we group the samples based on their RTT with a group range of 10 milliseconds such that a series, $\{(\text{RTT}_i, N_i),\ i \geq 1\}$, is formed, where RTT_i is the middle 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 the RTT. The method reports a bot if the τ statistic is statistically greater or equal to zero, and a human player otherwise.

The detection rule is as follows. First, compute τ_1 and τ_2, the statistics of the Mann-Kendall test, for $\{(\text{RTT}_i, N_i),\ i \geq 1\}$ and $\{(\text{RTT}_i, N_i),\ i \geq 2\}$, respectively. If $\text{sign}(\tau_1) * \text{sign}(\tau_2) > 0$, that is, the trend remains the same regardless of the first RTT group, the identification is completed by $is.bot \leftarrow I(\tau_1 \geq 0)$. Otherwise, we compute τ_3 for $\{(\text{RTT}_i, N_i),\ i \geq 3\}$, and conclude the test by the sign of τ_3, that is, $is.bot \leftarrow I(\tau_3 \geq 0)$.

The classification result using the above procedures is plotted in Figure 17. Although most player traces are correctly judged, the scheme seems to have problems correctly identifying *DreamRO* bots. We find that *DreamRO* traces sometimes exhibit a negative correlation between the RTT and the packet rate, which we characterize as human behavior. The reasons for this behavior need further investigation. However, we consider such methods based on human behavior rather interesting and potentially useful. An analysis of the relationship between game traffic patterns and user behavior will be part of our future work.

7. Performance Evaluation

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

The evaluation results demonstrate that the first two strategies, *command timing* and *burstiness trend*, perform rather well, as shown in Figure 18. Specifically, both methods yield correct decision rates higher than 95% and false negative rates lower than 5%, given an input greater than

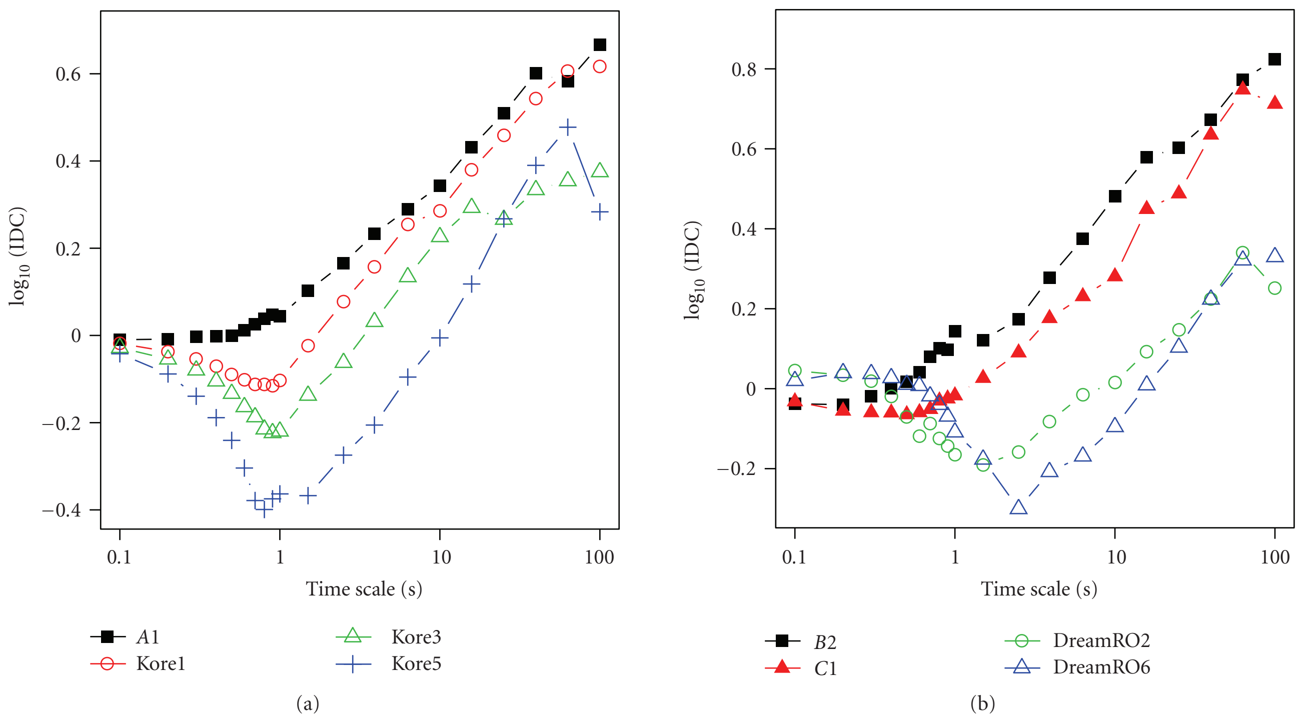


Figure 10: IDC plots for different traces. Player traces are represented by filled symbols; bot traces are represented by unfilled symbols. Note the trend of IDC for bot traces has a "dip" around 1 second.

10000 packets. From the viewpoint of business operations, a bot detection solution 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, but misjudging a bot (as a human player) should 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 decision 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 based on an integrated approach as the ensemble learning approach in the machine learning field, that is, by applying multiple schemes simultaneously and combining their results according to desired preference. For example, if a conservative judgement is preferred, a traffic stream would only be deemed to correspond to a game bot if all schemes agree with that decision. 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. This corresponds to combining two individual classifiers in parallel, using a static and nontrainable combiner with an ensemble learning terminology.

The performance of these integrated classifiers is rather good in terms of reducing the occurrence of certain kinds of false alarm, as illustrated in Figure 19. The conservative approach reduces the false positive rate to zero and achieves a 90% correct decision rate, given an input size of 10000 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 2000 packets.

8. Discussion

In this section, we first discuss the generality of our proposed schemes, that is, whether they can be generalized to other bot series designed for *Ragnarok Online* or other games, without considering counter-attacks. We then evaluate the robustness of the schemes under the presence of counter-strategies from bot developers. Finally, we consider the issues related to server-side deployment and how to further improve the detection accuracy with reactive strategies.

8.1. Generality of Proposed Detection Strategies. As bots are not actually "watching" screens, they perceive the environment by analyzing the information conveyed by server packets. Thus, the design naturally leads to the situation that, whenever a bot is aware of a change in the game world, it will *react* by sending commands back to the server. When a succession of commands, rather than a single command, needs to be sent, to avoid overwhelming the network and the server, some pacing mechanism that spreads the release time of the commands would be used. By this reasoning, we argue that the regularity in client response times is not unique to the particular bots we studied, but commonly exists in MMORPG bots. In other words, the *command*

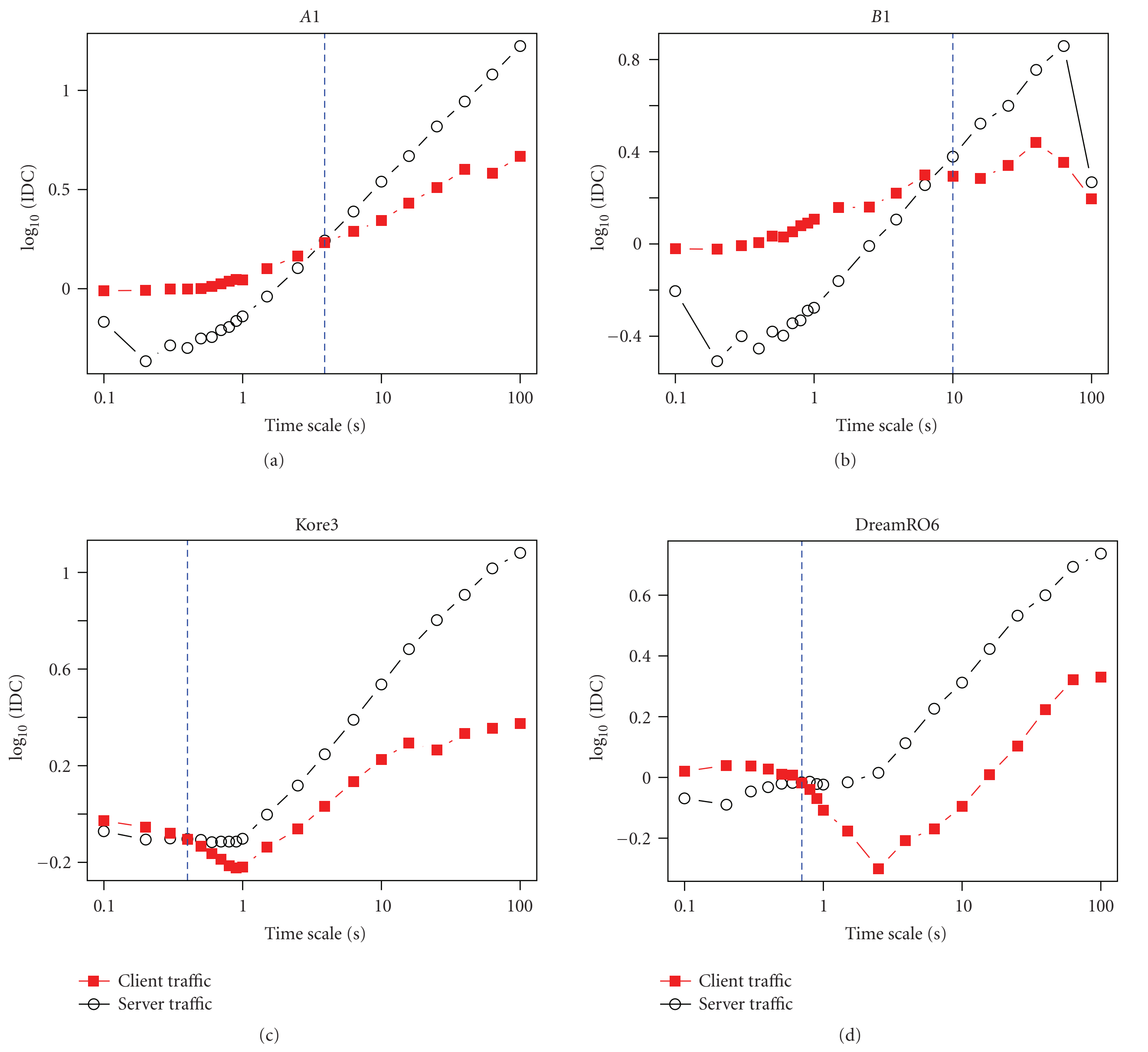


Figure 11: IDC magnitude comparison. The IDC of client packet arrivals versus the IDC of server packet arrivals.

timing scheme is generalizable as it is based on the fact that bots react to server packets with a certain form of regularity.

In view of traffic burstiness, as bots react based on server packets, which are inevitably periodic to ensure smooth screen updates, the periodicity will be *propagated* into the bot traffic. Therefore, we will definitely find lower burstiness in time scales around the status update frequency or the period required to process a command or response. In addition, since bots do not exhibit human-like behavior, such as *heavy-tailed activities* [12], the large-scale burstiness of bot traffic will be lower than that of traffic generated by human players. This explains why the patterns we observed in multiscale analysis of traffic burstiness (Section 5.3) should be generally observable, rather than being a particular phenomenon in our settings.

The sensitivity to network conditions should be game independent because it reflects user reaction to the rate of screen updates, 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, which is usually unavoidable when scheduling a series of successive actions. Thus, unlike human players, the command sending rate of bots would not correlate significantly with the pace of screen updates.

8.2. Robustness Against Random-Delay Counter-Attacks. Our traffic analysis approach is particularly generalizable because it is independent of game design and content; however, one of the biggest challenges to our schemes could be

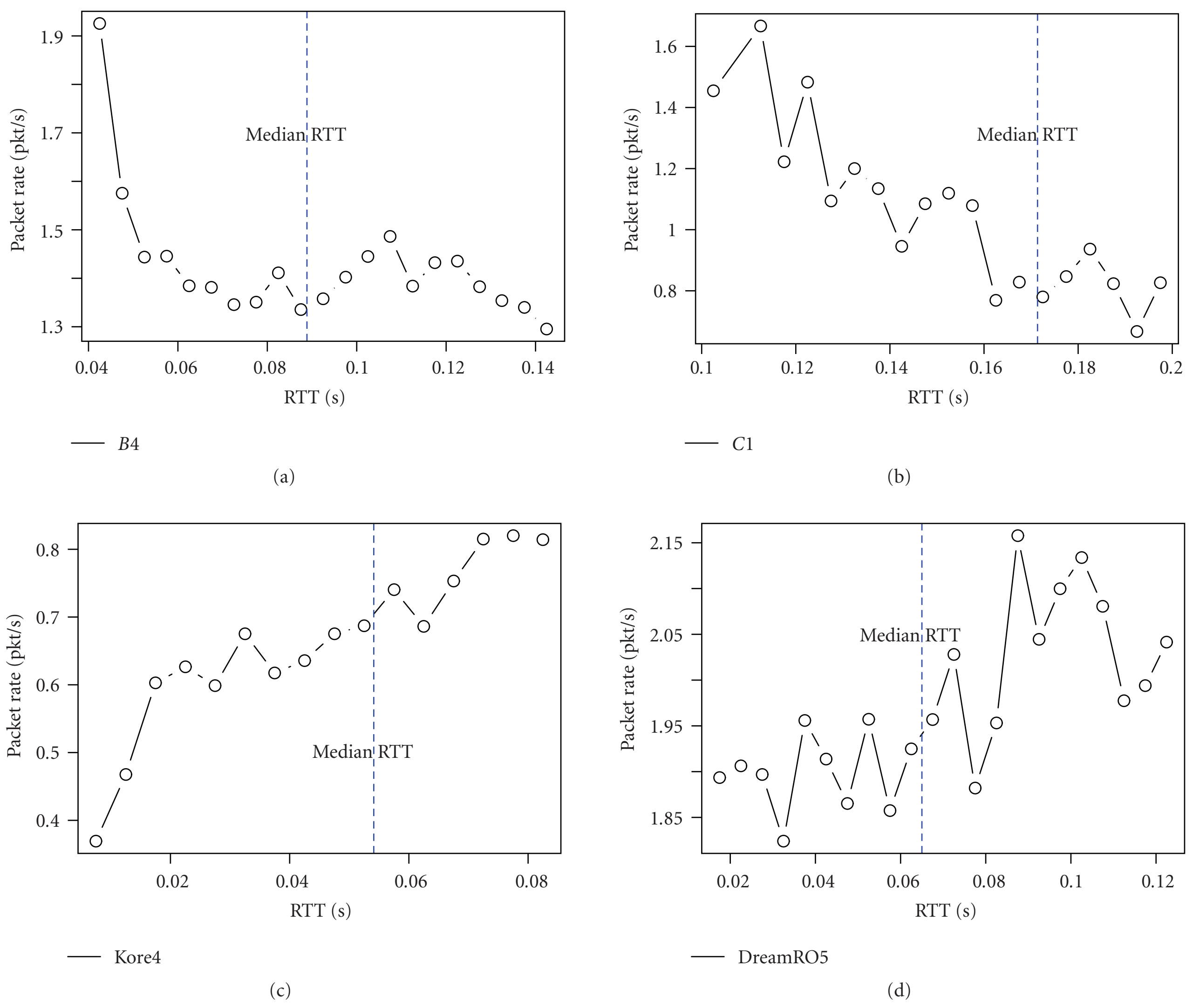


Figure 12: Average packet rates versus round trip times plot. The figures exhibit a downward trend for player traces, and an upward trend for bot traces.

their robustness under counter-attacks from bot developers. Since our strategies use packet timestamps as the only input, an obvious counter strategy would be to *add random delays to the release time of client commands*. Because the *command timing* scheme relies on the regularity of bot behavior, it is inevitable that random delays would make less effective. However, we argue that the schemes based on traffic burstiness and human reaction to network conditions are resistant to such attacks.

The *burstiness trend* scheme is immune to random-delay attacks because bots must always take actions based on the up-to-the-minute information that is sent from game servers periodically. Adding random delay to the client response time would not affect the regularity unless the added delay is longer than the status update intervals by *orders of magnitude* or it is *heavy-tailed*. However, adding such long delays would make the bots less threatening, as we explain in the next subsection (see Section 8.3).

We demonstrate this robustness property by simulations. Using the *Kore1* trace as an example, we postpone the release of each command by random delays drawn from uniform and exponential distributions, respectively. The IDCs of the original packet arrival process and those of the intentionally-delayed versions are shown in Figure 20. It is clear that random delays do not remove the “dip” from the burstiness trend; they only mitigate the extent of or change the location of the dip. The figure also shows that the *burstiness magnitude* scheme is not only resistant to simple random-delay attacks; it is also more effective in terms of detection capability because the burstiness of randomly-delayed traffic is even lower than the original.

Moreover, random delays do not have any effect on the *pacing* scheme if they are independent and identically-distributed. In this case, delays merely increase the variance of the command rates, but they do not change the average command rates.

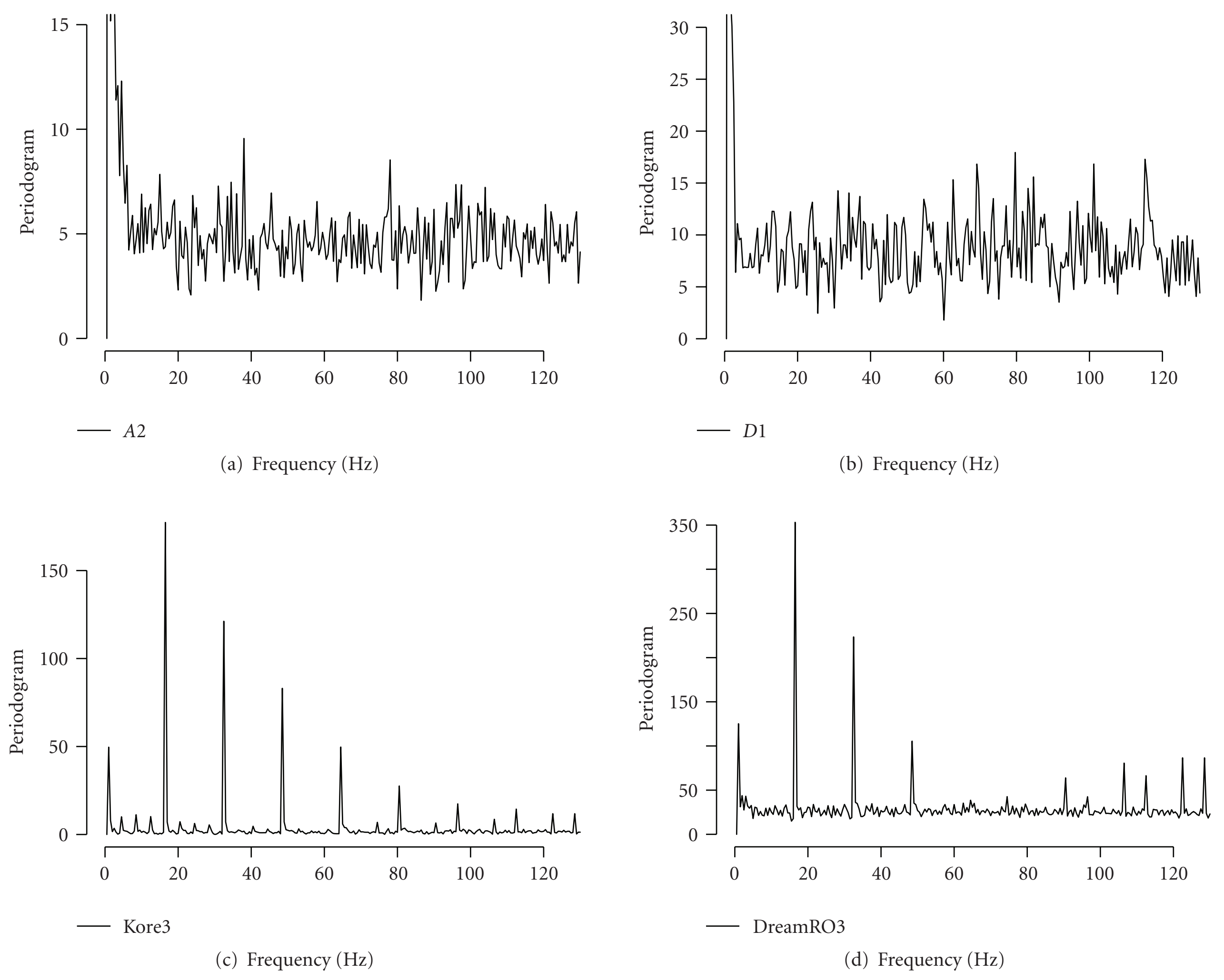


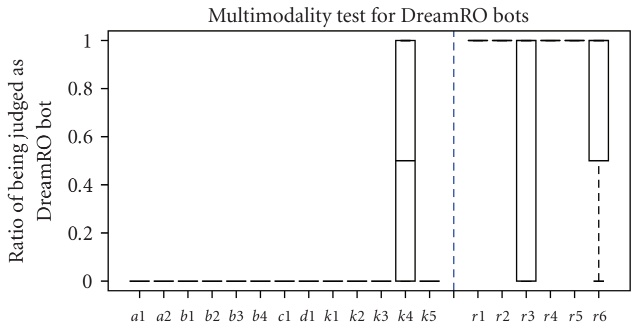
FIGURE 13: Periodograms of corresponding histograms (of client response times) in Figure 8. Note strong frequency components exist in bot traces.

8.3. Sophisticated Counter-Attacks. If bots are constantly interrupted by the game operator, their developers would naturally try to improve their programs to hide from the detection schemes. Since our proposed schemes are based on the discriminability of traffic between bots and human players, intuitively, designing bots that better resemble human players would lead to more sophisticated counter-attacks. Next, we discuss three possible counter-attacks that bots may adopt, their effectiveness on the proposed schemes, and their consequent weakness (if applicable).

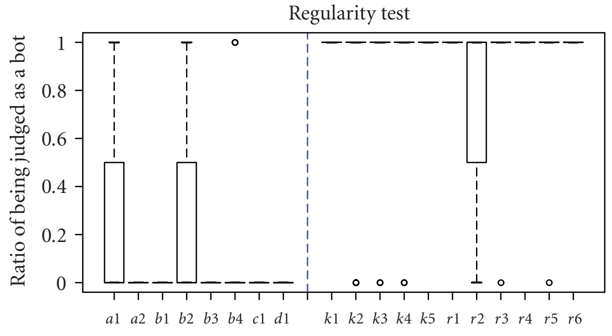
8.3.1. Heavy-Tailed Random Delays. This attack is similar to the simple random delay attack we discussed in the last subsection, except that now the delay times are drawn from a heavy-tailed distribution instead. This type of attack would make the *burstiness trend* scheme less effective as the "dip" effect on the burstiness trend is less significant. Similarly, this attack would be effective against the *burstiness magnitude* scheme, since the heavy-tailed delays would significantly raise the variability of client traffic in multiple time scales so that the the burstiness of client traffic and server traffic would be comparable.

As a demonstration, we simulate the effect of this attack. The simulation results are plotted as the delayed Pareto series, which draws random delays from a Pareto distribution with the shape parameter 1.2, in Figure 21. We observe that the attack makes the "dip" effect insignificant, which demonstrates the effectiveness of this *heavy-tailed delays* attack.

Weakness of this Counter-Attack. In trying to mimic human activities, however, the heavy-tailed ON/OFF delays would make bots much less efficient at reaping rewards because of the nonsignificant probability of long idle times. Taking the delayed Pareto series as an example, to have similar burstiness to the original series in large time scales, the total time needed to achieve the same tasks is approximately four times greater than the original. In other words, a bot followed the delayed Pareto random delays would be four times less effective than it could be. Thus, even though bot developers



(a) Trace



(b) Trace

Figure 14: Bot identification results using the *command timing* method which comprises the multimodality test and the regularity test.

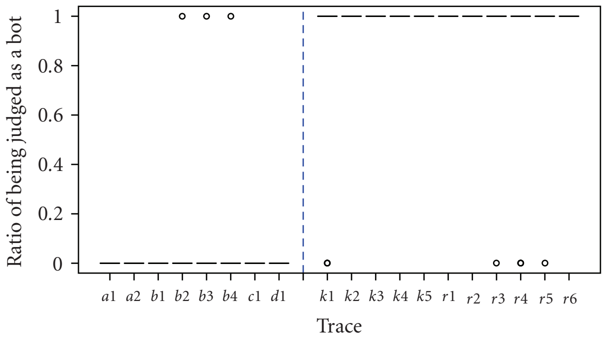


Figure 15: Bot identification results using the *burstiness trend* method.

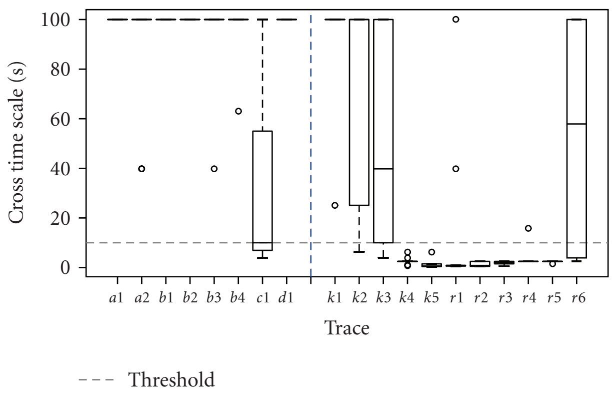


Figure 16: Bot identification results using the *burstiness magnitude* method.

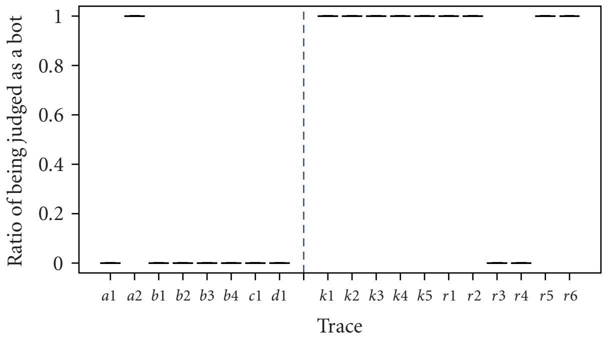


Figure 17: Bot identification results using the *pacing* method.

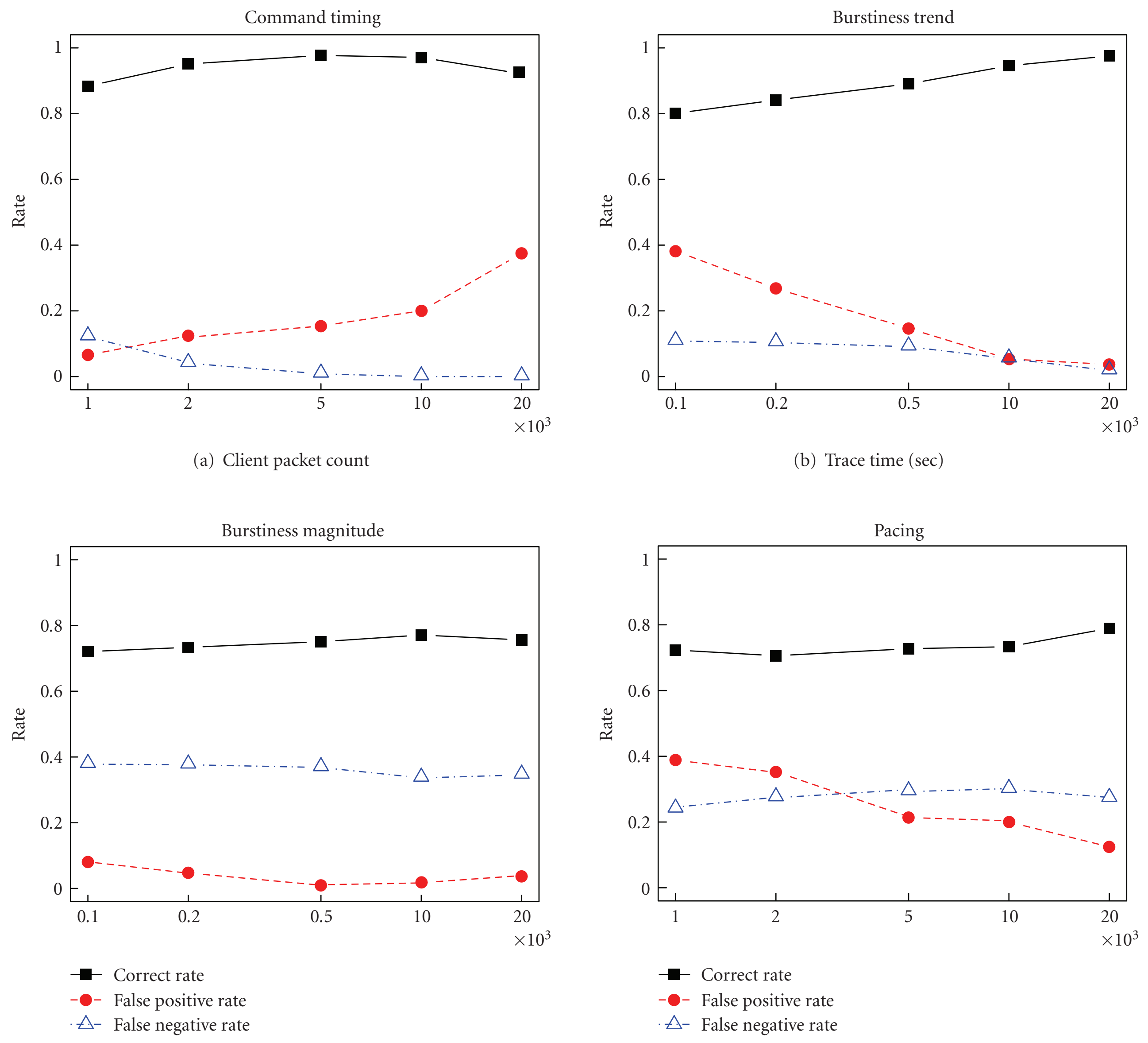
can fool detection schemes by incorporating heavy-tailed ON/OFF activities, the bots will pose a much smaller threat to the balance of the game world.

8.3.2. Independent Command Release Time. With this attack, bots have to issue commands according to a schedule that is independent of the server packet arrival time. This attack would make the *burstiness trend* scheme ineffective, as the client traffic no longer contains the regularity inherited from server traffic. As shown in Figure 21, the Exp(1) and Pareto series simulate the effect of this counter-attack, where the client packet release time is completely decided by the value drawn from an exponential distribution (with an average of 1 second) and a Pareto distribution (with the shape parameter 1.8). Under this attack the "dip" effect may completely disappear so that the *burstiness trend* scheme would be now unable to differentiate bots from human players.

Weakness of this Counter-Attack. If a bot only sends out packets when one or more commands are ready, the client packet departure time would correlate with server packet arrival time, as bot commands are decided based on the up-to-date game states conveyed by server packets.

Thus, to make the client packet departure time completely independent of server packet arrivals, a bot should decide when to send out packets by a different schedule. Whenever the scheduled timer is triggered, the bot must issue a command no matter whether the command is necessary or not. Consequently, there must be some cases where the bot is not ready to issue any new commands, but it must send out some because new game states have not been received or processed yet, or there is nothing else to do at that point. Hence, the bot would have to perform certain actions that are not unnecessary, such as moving around or making an insignificant gesture. In such cases, bots that perform meaningfulness actions would be detected more easily by higher-level bot detection schemes equipped with knowledge about game semantics.

8.3.3. Intentional Packet Pacing. This attack intentionally adapts the packet sending rate to the measured network



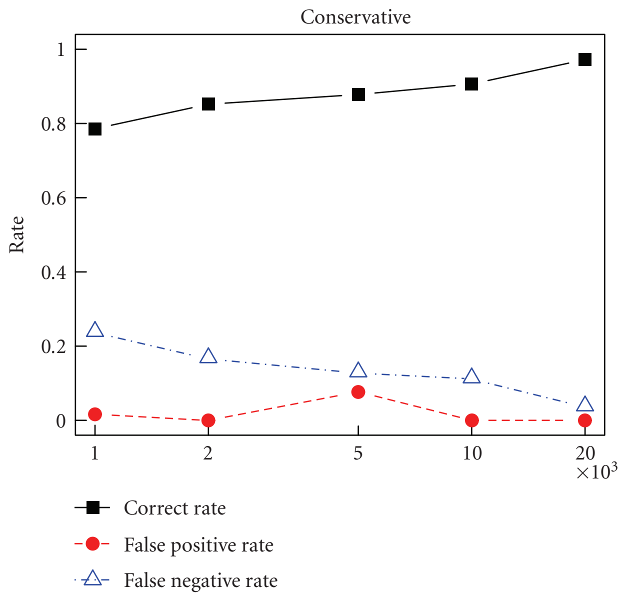
(a) Client packet count (b) Trace time (sec) (c) Trace time (sec) (d) Number of RTT samples

Figure 18: Evaluation results for the proposed decision schemes with different input size.

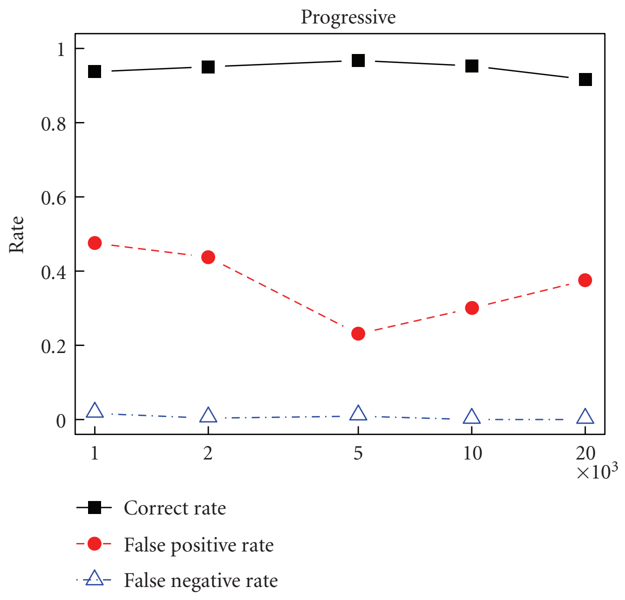
conditions. The *pacing* scheme would be fooled, since bots can simulate human players' sensitivity to network conditions in terms of the command sending rate.

Extension of the Pacing Scheme. On the positive side, the *pacing* scheme can be further extended to model players' general behavior under different network conditions. For example, if the network lags are serious, human players could not perform as well as they would with mild lags, but bots can. Similarly, players normally cannot tolerate poor network quality continuously for a few hours, but bots can. Although bots can simulate human sensitivity to network quality, we believe that certain kinds of player behavior are much more difficult to simulate. At the very least, the bots would have to pay for their imitation behavior in terms of efficiency (such as pretending to miss a target, or leaving the game prematurely due to serious lags). Paired with reactive identification (which we will introduce in Section 8.5) and application-level information, we believe the human behavior approach is still a promising way to distinguish "fake" human players from "genuine" human players.

8.4. Server-Side Deployment. In practice, the bot detection mechanisms should be implemented at the server side because client-side software can always be compromised by crackers. We now discuss whether our schemes, which are based on client traffic traces, would be ineffective if they were run at the server side.



(a) Client packet count



(b) Client packet count

FIGURE 19: Evaluation results for the integrated schemes with different input size.

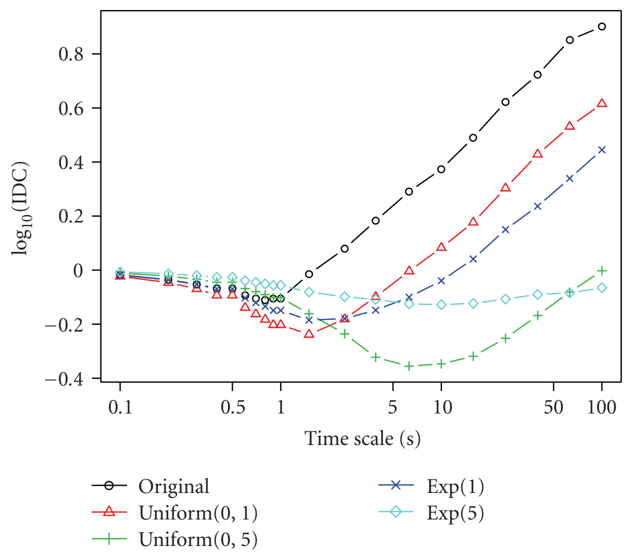


FIGURE 20: The IDC of the original packet arrival process in *Kore1* and those of the intentionally-delayed versions.

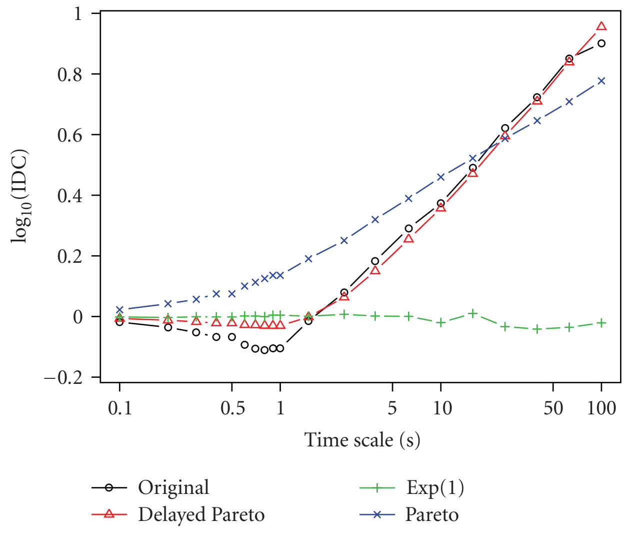


FIGURE 21: The IDC of the original packet arrival process in *Kore1* and those of the versions under sophisticated counter-attacks.

Coincidentally, the packet timestamps collected at the server side can be seen a random-delay-augmented version of the packet timestamps collected at the client side, where the random delays are caused by queueing fluctuations along the network path from the client to the server. Furthermore, the queueing variations of a typical Internet path is much less than one second in most cases, so the added random delay is considerably less than the time scales we are concerned about. Therefore, we can directly apply the discussion in the preceding section (Section 8.2) here. That is, although the *command timing* scheme would be made ineffective by random delays, the strategies based on traffic burstiness and users' sensitivity to network conditions will continue to be effective when they are deployed on the game servers.

8.5. Reactive Identification. In this paper, all the bot detection strategies introduced so far are purely passive, that is, the schemes only make decisions based on *observation* of the packets flowing from a game client to a server and vice versa. We argue that the schemes can be extended to be *reactive* to improve the efficiency and correctness of the decisions.

For example, in the pacing scheme, we determine whether a traffic stream belongs to a bot by its adaptation the packet rates of different network conditions. If the network conditions are quite stable, this scheme would be less effective because samples would be not diverse enough to examine the client's behavior with different network delays. To solve this problem, the detection scheme can purposely alter the state update intervals for a suspicious character (detected by other detection strategies or by user reports), so that it can collect more information about how the client adapts to a particular game pace more quickly and reliably.

In another example of reactive identification, a character is enticed to perform certain actions that tend to draw out regularity. For instance, the server can entice a suspicious character to pursuer a monster by offering an attractive reward, and then check if the character is issuing movement commands that are highly periodic or dependent on server packet arrival times. Since making false accusations against bona fide players should be avoided as much as possible, such reactive identification schemes could be used as a second line of defense for more accurate detection of game bots.

9. Future Directions

Like the competition between computer virus writers and antivirus software developers, the competition between bot developers and bot detection mechanisms will never end. Moreover, as writing game bots is usually rewarding in terms of real money (many game bots are commercial with a time-limited license), bot developers will certainly continue trying to make their programs undetectable by any bot identification algorithm. The pure traffic-based detection schemes we proposed are generalizable because they are independent of the game design and semantics, but they might be deceived by sophisticated counter-attacks that mimic human gaming activities. However, we do not think this is the end of the campaign against bots. Many tools are still available for use with traffic-based schemes to tackle game bots with more robustness and efficiency. In the following, we detail some promising strategies that we believe will help in the automatic bot detection problem.

9.1. Players' General Behavior. Even though bots try to mimic how human players control their virtual characters, certain aspects of human behavior are difficult to simulate with a computer program. For example, to make the route computation tractable in real time, the movements of virtual characters, decided by game logic or bot programs, are nearly all computed by using the $A^\star$ algorithm [21]. The computer-decided trajectory, however, is very different from the trajectory of a virtual character controlled by a human. How to generate a human-like path that simultaneously considers the tasks at hand, the character's status, and the environment (enemies, terrain, obstacles, etc.), is an issue that has yet to be resolved and is thus an ongoing research topic in the field of artificial intelligence [22].

In sum, our strategy is to model certain types of human behavior that current AI techniques cannot imitate well (such as movement trajectories [8, 9]), and detect game bots based on the results as bot programs cannot mimic such behavior well.

9.2. Change of Player Behavior under Various Network Conditions. Online gaming experiences are strongly related to the QoS of the network path, including the network delay, jitter, and packet loss rate [23–26]. When the network quality is poor, the interactivity and responsiveness of game play will be degraded, and players will have difficulty controlling their characters in a timely and accurate manner. Hence, they may become less involved in the game world, feel frustrated, or even be angry so that their mood may further degrade their gaming performance. Externally, human players behave differently in terms of their gaming performance in game, their typing speed, the way they control the game characters (with a keyboard or mouse), and the way they treat other virtual players or nonplayer characters. More importantly, such behavioral changes due to different network QoS conditions should vary among players, that is, the changes are unique to each player.

Our strategy puts game bots in a dilemma by employing this property. On one hand, bots must mimic players' changeable behavior under various network conditions in order to be human-like, which is not an easy task in the first place. On the other hand, the bot will be easily detectable if there are a number of bot instances (i.e., more than one character is controlled by the same bot program) running in a game, since their "sensitivity" to network quality would be very similar. Of course a sophisticated bot may provide a number of profiles to simulate different "personalities" with different bot instances, but the number of profiles it could provide would be limited. Thus, a bot program would eventually be detected by a "personality profile" that multiple bot instances use simultaneously.

9.3. Interrelationships of Characters. In some of MMORPGs, (http:// en.wikipedia.org/wiki/Real-money_trading) real-money trading (RMT) is prevalent, so game bots are used by some people to make a profit by selling virtual currency and goods to other gamers. People who just want to make money rather than play the game are usually referred to as "gold farmers" or simply "farmers." To gather valuable in-game resources quickly, a farmer usually runs dozens of bots simultaneously, which form a number of groups. The groups of bots have the ability to collaborate with each other, so they can move and act together to form a powerful force. Groups of bots can cover each other by sharing needed items, healing, or mutual shielding. They may also adopt a division of labor strategy whereby some do the fighting and others are responsible for collecting the loot.

By analyzing the behavior of game characters, the bot groups run by farmers can be detected by the following characteristics (1) characters run by game clients that are located at the same LAN, (2) they usually participate in and leave the game at approximately the same time, (3) they usually move and act together all the time, (4) they do not interact with characters played by other gamers unless necessary, and (5) they frequently exchange items or virtual currency with other characters in the same group.

We believe that a group of characters with all the above characteristics is probably controlled by game bots. This scheme can serve as an efficient way to identify suspected gold farmers. They can then be examined in more detail with more accurate identification schemes.

9.4. Collective Decisions. As game bots tend to monopolize resources in a game and break the balance of the game world, most legitimate players would be happy if bots could be eliminated from games completely. Being overwhelmed by users' complaints, the most common strategy currently adopted by game companies is that, whenever a character is reported as a bot, a game master must take some time to follow and observe the suspect, until he/she can judge whether the report is factual or not. However, this method is very inefficient as it takes a great deal of time to manually identify whether a character is controlled by a program or a human, and the number of game masters is also limited. In addition, more advanced bots may temporarily leave the game when they detect the presence of game masters around their characters, and return later.

The strategy we propose relies on users' reports to decide whether a character is a game bot. The rationale is that legitimate players usually have strong incentives to report bot use. We argue that if user reports can be appropriately aggregated, they would form a powerful weapon against bots. It would be not difficult to design a mechanism that allows a player to report a bot-controlled character, or, conversely, a witness of a human-controlled character. However, such mechanisms might have the following problems [27].

(1) Misjudgement: players may mistake a normal player for a bot, or vice versa.

(2) Ballot stuffing: a bot owner may collude with other bot owners to lodge fake reports in order to avoid detection by the system.

(3) Bad mouthing: a human player might be targeted by a group of players who falsely accuse him/her of being a bot owner.

The main challenge of this strategy is how to detect incorrect and false reports. Even though individual reports might be intentionally or unintentionally incorrect, if incorrect ones could be detected and removed automatically, we could determine whether a character is bot-controlled with the help of the game's participants.

9.5. Honey Pots. While CAPTCHA tests [11] might be the only sure way to distinguish between human beings and computer programs, forcing players to conduct such tests is not appropriate in many games, since the tests inevitably interrupt the flow of game play. We consider that honey pots, which are also be based on human intelligence like CAPTCHA but they are less intrusive, would be more appropriate for detecting game bots.

For example, the game designer may put a special-purpose monster in a game. The monster would be exactly like the other monsters, except that it has a banner "*Do not attack me unless you are a bot!!*" above its head, and the banner text is distorted so that it is not easily recognized by optical character recognition (OCR) techniques. In this way, if a player keeps slashing the monster, we would know that the player is likely a bot program. Even though the honey pot mechanism should be very effective in capturing bots, a player may still attack the honey-pot monster by mistake or unintentionally ignore the warning message. Hence, it would be better to use this strategy in cooperation with other schemes to derive more accurate decisions about bot use.

10. Conclusion

Automatic game bot identification is a new and interesting topic that involves networking, artificial intelligence, psychology, human-computer interaction, social networking, and game design. In this paper, we have addressed the game bot problem and proposed a number of methods that identify game bots automatically using a traffic analysis approach. Taking *Ragnarok Online* as a case study, we obtained and analyzed packet traces for human players and mainstream game bots under different network settings. We have shown that the traffic corresponding to bots and human players is distinguishable in various respects, such as the regularity in client response times, the trend and magnitude of traffic burstiness in multiple time scales, and user sensitivity to network conditions.

Based on the traffic patterns identified, we have proposed four general decision strategies and two integrated schemes for bot detection. 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 10000 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 2000 packets. We have shown that the proposed methods are generalizable to other bot series and games and robust against simple random-delay counter-measures from bot developers. In addition, we have discussed the issues regarding deployment and reactive detection of bots.

Due to the highly profitable nature of game bots, bot developers will try anything to improve their programs so that they are undetectable by any bot identification algorithm. The pure traffic-based detection schemes we propose are generalizable because they are independent of game design and semantics; however, they might be deceived by sophisticated counter-measures that mimic human gaming activities. This situation is not completely avoidable because game bots can always imitate human activities at the network level.

Given the intrinsic difficulty of the bot identification problem, that is, telling computers and humans *passively* while human behavior can be highly heterogeneous and variable, we believe that the most effective bot detection scheme should be *multimodal* rather than a single mode approach. To this end, we have explored a number of promising strategies (1) exploit bots' inability to imitate human behavior, (2) exploit bots' insensitivity to the changing network conditions, (3) exploit the interrelationships of bot-controlled characters, (4) aggregate user reports intelligently, and (5) use honey pots that only human players can avoid getting trapped by. We are currently investigating this array of detection methods and applying them in practical ways. We hope that this *nonintrusive* and *game-independent* bot detection study will help increase awareness of the increasingly serious bot problem in the online game community.

Acknowledgments

The authors would like to thank the editors and the anonymous reviewers for their helpful comments. This work was supported in part by Taiwan Information Security Center (TWISC), National Science Council under Grants NSC 97-2219-E-001-001 and NSC 97-2219-E-011-006. It was also supported in part by Taiwan E-Learning and Digital Archives Program (TELDAP), National Science Council under Grants NSC 96-3113-H-001-010 and NSC 96-3113-H-001-012.

References

[1] B. S. Woodcock, "An analysis of MMOG subscription growth—version 21.0," http://www.mmogchart.com.

[2] N. E. Baughman and B. N. Levine, "Cheat-proof playout for centralized and distributed online games," in *Proceedings of the 20th Annual Joint Conference on the IEEE Computer and Communications Societies (INFOCOM '01)*, vol. 1, pp. 104–113, Anchorage, Alaska, USA, April 2001.

[3] E. Cronin, B. Filstrup, and S. Jamin, "Cheat-proofing dead reckoned multiplayer games," in *Proceedings of the 2nd International Conference on Application and Development of Computer Games (ADCOG '03)*, Hong Kong, January 2003.

[4] M. DeLap, B. Knutsson, H. Lu, et al., "Is runtime verification applicable to cheat detection?" in *Proceedings of the 3rd ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '04)*, pp. 134–138, ACM Press, Portland, Ore, USA, August-September 2004.

[5] J. Yan and B. Randell, "A systematic classification of cheating in online games," in *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games (NetGames '05)*, pp. 1–9, ACM Press, Hawthorne, NY, USA, October 2005.

[6] S. F. Yeung, J. C. S. Lui, J. Liu, and J. Yan, "Detecting cheaters for multiplayer games: theory, design and implementation," in *Proceedings of the 2nd IEEE International Workshop on Networking Issues in Multimedia Entertainment (NIME '06)*, vol. 2, pp. 1178–1182, Las Vegas, Nev, USA, January 2006.

[7] H. Kim, S. Hong, and J. Kim, "Detection of auto programs for MMORPGs," in *Proceedings of the 18th Australian Joint Conference on Artificial Intelligence (AI '05)*, pp. 1281–1284, Sydney, Australia, December 2005.

[8] K.-T. Chen, A. Liao, H.-K. K. Pao, and H.-H. Chu, "Game bot detection based on avatar trajetory," in *Proceedings of the 7th IFIP International Conference on Entertainment Computing (ICEC '08)*, Pittsburgh, Pa, USA, September 2008.

[9] K.-T. Chen, H.-K. K. Pao, and H.-C. Chang, "Game bot identification based on manifold learning," in *Proceedings of the 7th Annual Workshop on Network and Systems Support for Games (NetGames '08)*, Worcester, Mass, USA, October 2008.

[10] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "CAPTCHA: using hard AI problems for security," in *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '03)*, pp. 294–311, Warsaw, Poland, May 2003.

[11] P. Golle and N. Ducheneaut, "Preventing bots from playing online games," *Computers in Entertainment*, vol. 3, no. 3, article 3, pp. 1–10, 2005.

[12] K.-T. Chen, P. Huang, and C.-L. Lei, "Game traffic analysis: an MMORPG perspective," *Computer Networks*, vol. 50, no. 16, pp. 3002–3023, 2006.

[13] R. Gusella, *A characterization of the variability of packet arrival processes in workstation networks*, Ph.D. dissertation, University of California at Berkeley, Berkeley, Calif, USA, 1990.

[14] R. Gusella, "Characterizing the variability of arrival processes with indexes of dispersion," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 2, pp. 203–211, 1991.

[15] J. A. Hartigan and P. M. Hartigan, "The dip test of unimodality," *Annals of Statistics*, vol. 13, no. 1, pp. 70–84, 1985.

[16] P. M. Hartigan, "Computation of the dip statistic to test for unimodality," *Applied Statistics*, vol. 34, no. 3, pp. 320–325, 1985.

[17] R. A. Fisher, "Tests of significance in harmonic analysis," *Proceedings of the Royal Society of London. Series A*, vol. 125, no. 796, pp. 54–59, 1929.

[18] W. A. Fuller, *Introduction to Statistical Time Series*, John Wiley amp; Sons, New York, NY, USA, 1996.

[19] P. Bloomfield, *Fourier Analysis of Time Series: An Introduction*, John Wiley
amp; Sons, New York, NY, USA, 2000.

[20] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1-2, pp. 81–93, 1938.

[21] R. Dechter and J. Pearl, "Generalized best-first search strategies and the optimality of A^*," *Journal of the ACM*, vol. 32, no. 3, pp. 505–536, 1985.

[22] B. Gorman and M. Humphrys, "Towards integrated imitation of strategic planning and motion modeling in interactive computer games," *Computers in Entertainment*, vol. 4, no. 4, article 10, pp. 1–14, 2006.

[23] K.-T. Chen, P. Huang, and C.-L. Lei, "Effect of network quality on player departure behavior in online games," to appear in *IEEE Transactions on Parallel and Distributed Systems*.

[24] K.-T. Chen, P. Huang, and C.-L. Lei, "How sensitive are online gamers to network quality?" *Communications of the ACM*, vol. 49, no. 11, pp. 34–38, 2006.

[25] K.-T. Chen, P. Huang, G.-S. Wang, C.-Y. Huango, and C.-L. Lei, "On the sensitivity of online game playing time to network QoS," in *Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM '06)*, pp. 1–12, Barcelona, Spain, April 2006.

[26] T.-Y. Huang, K.-T. Chen, P. Huang, and C.-L. Lei, "A generalizable methodology for quantifying user satisfaction," *IEICE Transactions on Communications*, vol. E91-B, no. 5, pp. 1260–1268, 2008.

[27] C. Dellarocas, "The digitization of word of mouth: promise and challenges of online feedback mechanisms," *Management Science*, vol. 49, no. 10, pp. 1407–1424, 2003.