# You Are a Game Bot!: Uncovering Game Bots in MMORPGs via Self-similarity in the Wild


Eunjo Lee
NCSOFT
gimmesilver@ncsoft.com

Jiyoung Woo
Korea University
jywoo@korea.ac.kr

Hyoungshick Kim
Sungkyunkwan University
hyoung@skku.edu

Aziz Mohaisen
SUNY Buffalo
mohaisen@buffalo.edu

Huy Kang Kim
Korea University
cenda@korea.ac.kr



***Abstract*—Game bots are a critical threat to Massively Multiplayer Online Role-Playing Games (MMORPGs) because they can seriously damage the reputation and in-game economy equilibrium of MMORPGs. Existing game bot detection techniques are not only generally sensitive to changes in game contents but also limited in detecting emerging bot patterns that were hitherto unknown. To overcome the limitation of learning bot patterns over time, we propose a framework that detects game bots through machine learning technique. The proposed framework utilizes self-similarity to effectively measure the frequency of repeated activities per player over time, which is an important clue to identifying bots. Consequently, we use real-world MMORPG ("`Lineage`", "`Aion`" and "`Blade & Soul`") datasets to evaluate the feasibility of the proposed framework. Our experimental results demonstrate that 1) self-similarity can be used as a general feature in various MMORPGs, 2) a detection model maintenance process with newly updated bot behaviors can be implemented, and 3) our bot detection framework is practicable.**


## I. Introduction

Worldwide, over 40% of Internet users play online games in their leisure time [13]. Massively Multiplayer Online Role-Playing Games (MMORPGs) are one of the most popular genres of such games. In an MMORPG, each player creates a character in the virtual game world to perform various activities including combating other players, socializing with other players and collecting game items. "Item-pay" is a primary business model that offers financial incentives for game developers [9]. In this model, users need to pay real money to purchase in-game items that can increase their characters' abilities in a game.

MMORPG developers expect users to (legally) purchase game items because their MMORPGs are generally designed to require users to invest a significant amount of human labor (i.e., playing time) to acquire those items. Invariably, as an MMORPG becomes more popular, it is likely to lead to the creation of black markets where illegally acquired items can be freely traded in third party marketplaces outside of the game world. Organized cyber criminals and malicious users have also started using a tool known as game bot to automatically collect valuable and rare game items in an economical manner.

A game bot is a computer program that plays games autonomously instead of human users, typically used for in-game cheating. Unsurprisingly, the use of game bots has become one of the most serious security threats to MMORPGs. This is in part because such a game tactic not only results in significant financial losses in the short term, but also leads to fairness and reputation damages in the long term; it causes a significant number of game users to churn, reduces the revenue from those subscription fees collected, influences the inflation rate and eventually triggers the collapse of an in-game economy. Game bots make honest users feel frustrated and dissatisfied with the game because they monopolize virtual resources and contents while honest users compete for limited resources and contents. Castronova [5] studied the monetary damage caused by game bots in World of Warcraft, an MMORPG developed by Blizzard, via various aspects including customer service cost, a technical cost for bot enforcement, and revenue's penalty. The indirect cost of game bots was estimated to be approximately 18 million USD per year.

To address those issues, game publishers currently invest significant efforts to develop solutions for mitigating game bot activities [17]. The most fundamental step is to identify game bots. Several data mining techniques have been developed by game publishers to distinguish game bots from (normal) human users. In those techniques, the feature construction and selection processes are critical to enhance the detection performance. However, such processes are often labor-intensive and depend greatly on game contents or play patterns that are genre-specific. Therefore, the efficacy of the resulting detection model inherently decreases over time because game developers update game contents periodically, and game bots dynamically adjust their behaviors to such updates to evade detection. In general, constructing a specialized detection model for each MMORPG and maintaining the constructed model over time are very challenging tasks for game publishers.

To overcome those problems, a more generic model is required without the dependency on specific game contents. Moreover, this detection model should be automatically maintained over time. To achieve these goals, we focus on the repetitive activities of game bots, which are typically found in in-game logs. Our investigations show that game bots frequently repeat certain routine activities activities that are significantly different from activities of human users. Consequently, using this analysis, we propose a new bot detection framework that utilizes a metric called "*self-similarity measure*" (see §IV).



NDSS '16, 21-24 February 2016, San Diego, CA, USA



http://dx.doi.org/10.14722/ndss.2016.23436

Self-similarity is a feature to show the similarity of user actions as a function of the time lag. It is designed for finding repetitive patterns, especially periodic patterns of the series of actions and their frequency. We particularly consider various actions rather than a single action such as moving pattern to provide a strong self-similarity measure that is highly robust to the changes of target games in their subsequent updates.

To demonstrate the feasibility of the proposed model, we use ground truth samples from real-world MMOPRGs (specifically "`Lineage`", "`Aion`" and "`Blade & Soul`") that are particularly popular in South Korea. To this end, the key contributions of this paper are as follows:

1) We analyze the characteristics of real-world game bots using various aspects of exploratory data analysis.
2) We propose self-similarity as a feature that can be applied universally to detecting bots in MMORPGs and demonstrate its effectiveness with real datasets.
3) We propose a bot detection framework that includes a detection model maintenance process. The proposed framework can be spontaneously evolved with dynamic changes in bot behaviors over time.
4) We implement the proposed framework and utilize it for "Lineage" in Feb. 2015 resulting in more than 15,000 bot users being banned by our system. Our system is being tested and will be applied soon to "Blade & Soul" and "Aion". To this best of our knowledge, this is the first field deployment of bot detection systems using data mining in large-scale game services.

The remainder of this paper is organized as follows. In §II, we explain various terminologies and background theories used in this paper. In §III, we describe the characteristics and statistics of bots via the exploratory data analysis. In §IV, we outline the proposed methodology for detecting game bots and maintaining the detection model. In §V, we demonstrate the bot detection process and evaluate the performance of the proposed model. In §VI, we discuss several important issues for the real deployment of the proposed framework. In §VII, we review related works. Finally, we summarize our findings and conclude in §VIII.

## II. Background

In this section, we present the basic knowledge necessary to understand the game bot detection problem in MMORPGs.

### A. Terminology

We first define some important terminologies we will use in the rest of this paper.

**Character.** In MMORPGs, a character is an entity (often called an avatar in the virtual world) that is controlled by a user. Characters own their unique features regarding appearance, level, virtual goods, etc. Interestingly, most MMORPGs allow users to create multiple characters on his or her account.

**Bot.** A game bot is a cheat program designed to play the game autonomously utilizing artificial intelligence. The program is usually made by malicious software developers who steal the source code from online game companies or analyze the game client program and network traffic between the game client and server through reverse engineering without the game company's permission. A bot program usually provides functions that a user can use to set up behavior rules in various situations (see Fig. 1).

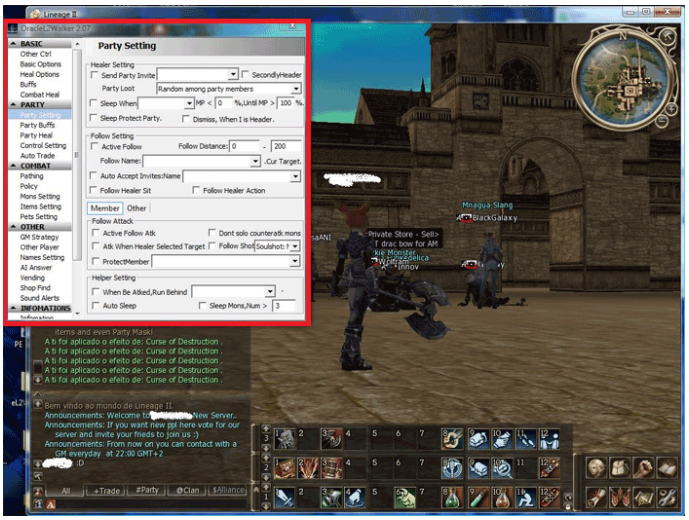



Fig. 1. Screenshot of a bot program. The main window in the background is a game client program for playing the game and the inset (in red box) is a bot program's window for configuring automated in-game bot actions.

**Gold farming group.** The main goal of a gold farming group (GFG) is to obtain cyber money from online games. In general, a GFG operates numerous machines and runs multiple client programs to minimize resource usage and cost. Most GFGs operate game bots or use cheap laborers to play the game.

**Game log.** Game logs record users' various activities and events that occur in the game world and also in the system. A log consists of an event identifier (id), time, character information and other context-related information. The event id is unique and depends on the event type. Logs have timestamps associated with them, indicating when activities or events occurred. In the games investigated in this paper, the time unit used is a millisecond. Character information encompasses the character's name, level, race, job, etc. Game logs have additional information that differs according to the event id. Such information is used to give detailed context information about an event. For example, for the game log of an event in which a character gives an item to another character, additional information would include the opponent's character information, information of the item that the character sends, a total number of items, and so on.

### B. Ecology of game world in MMORPGs

In the world of MMORPGs, a user controls a character, which is an avatar with a unique visual appearance and skills. MMORPGs provide a lifelike environment with a rich set of realistic user action types. Users explore various areas of the game world, hunt monsters, and complete quests (*i.e.*, given tasks) to earn experience points or goods that make themselves more powerful. Users can interact with other players in the game world as well as non-player characters (NPCs), which

are artificial intelligence agents controlled by a game server. Users can often collaborate or compete for their benefits. For example, players are campaigning as a party for a short period to complete a challenging quest for a single player. Furthermore, they can organize a guild for long-term collaboration. In general, many MMORPGs provide contents that only guild members can participate in (*e.g.*, battles between opposing guild members), and guild members are often enormously rewarded from those contents.

There are various economic activities in most MMORPGs. Users can trade virtual goods with each other in a game world, and they can even manufacture items directly. For example, there is a craft workshop in Aion, so users can make an item instead of obtaining it through hunting monsters or completing quests. Wealth is very important because expensive items are typically more powerful or useful, which is similar to the real-world economy. Users can gather valuable game items by buying them from an online store or other users.

While users simply pursue the fulfillment of a fixed goal in package games, they do not only want to enjoy the contents provided but also to have a more significant presence than other characters in online games [4]. Consequently, most online games encourage users to compete for limited resources by adjusting the probability of generating items because nobody is satisfied with an equal distribution. However as the period of game service increases so does the total wealth in the game world. Thus, it is necessary for inflation to occur with age to a greater or lesser degree [6]. If the inflation is very severe, the economic balance will collapse, and users will give up the game. Therefore, game developers design balance into the game and update new contents periodically to reduce inflation.

### C. Game bots and RMT

Game developers have designed MMOPRGs so that players have to take pre-scheduled courses to achieve high-level characters and become rich with cyber assets. That is, users should spend much time to complete those courses by engaging in repetitive play. In the case of Lineage, for example, game designers originally designed users to spend three hours per day for 3.5 years to reach a rank of the best players. This is a long period. Therefore, some users began to carry out various forms of cheating to level up their characters and acquire valuable cyber assets in a short time without sufficient effort and time investment.

Cheating in online games is no longer a private matter limited to individual players. Other players sustain damage because online games maintain a massive number of players and form social activities between players beyond a single player game. One of the most prevalent tools for cheating in online games is the game bot. Game bots enable users to cheat in a convenient way by automatically performing needed actions. An autonomous program that plays the game instead of a human is a typical type of game bot. Some users are eager to achieve a higher level in a short time by paying real money to obtain virtual goods (*e.g.* armor, weapon, *etc*.).

Users obtain game items and currency through a game play and can monetize that into real money. For real money trading (RMT), some players do various types of cheating forbidden by game companies. The connection between the virtual economy and the real economy drives virtual crime in online games and finally lead to the formation of illegal groups (so-called GFGs) that gather virtual goods in online games primarily for real money. For more efficient money gathering, GFGs largely use game bots. In addition, RMT is a frequently used method for tax evasion [11].

## III. Game bot characteristics

### A. Exploratory data analysis

We identified the characteristics of bots via the exploratory data analysis. To achieve this goal, we acquired a list of banned users from "Lineage", "Aion" and "Blade & Soul (B&S)". These games had serious bot problems; as a result, they made it a policy to ban bot programs and block their access permanently. We presented the statistics of three games in Table I. We performed a comparative analysis between bots and humans using various pieces of data, including demographic statistics, activity statistics, sequence, and network patterns.

TABLE I. A BRIEF SUMMARY OF GAMES.

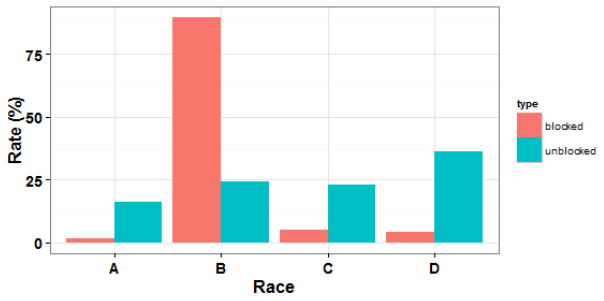
| Feature | Lineage | Aion | B&S |
|---|---|---|---|
| Release year | 1997 | 2008 | 2012 |
| Daily active users | 300K | 200K | 100K |
| Concurrent users | 150K | 80K | 50K |

*1) Demographic data:* Game characters have demographic characteristics similar to humans in the real-world, such as job, race, and gender. The features of bots differ from those of humans. Fig. 2 illustrates several demographic data differences. We found the significant differences between blocked and unblocked characters for both job and race. However, we believe the distinction is actually owing to just the characteristics of a job, not a race, because a race has a job constraint in Blade & Soul, *e.g.*, a user can choose only race B for job D.

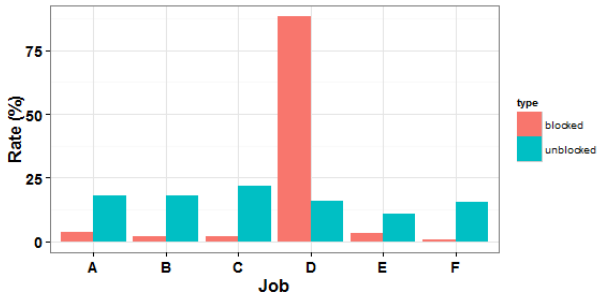
There are different combat skills required for each job, so a user can choose a job and enjoy different play patterns according to their goals and preferences. In general, game developers try to adjust the attributes of jobs and provide different pros and cons for each job. Consequently, the jobs that human users choose are evenly distributed because normal users tend to have separate goals and preferences. However, bots tend to choose a specific job in order to optimize their productivity. In Blade & Soul, job D, which blocked users chose the most, has benefits for hunting monsters and a disadvantage to PvP (Players versus Players; a type of combat within a game between two or more players). Besides, limited skills are allowed in job D, so this job is tedious. Accordingly, while bot programs can reduce the burden of control if they choose job D, normal users may feel bored by choosing it for the same reason.

Such pattern is maintained across different games, although with varying qualities. For example, Aion has different distributions of jobs, which show this trend in part; however, it is not as clear as in Blade & Soul (see Fig. 3).

*2) Activity statistics:* Bot characters can conduct activities continuously for a long time. Consequently, the frequency of activities bots do is significantly higher than the frequency of activities human users do. Fig. 4 shows that blocked users play longer and conduct more activities than normal users.




(a) race




(b) job

Fig. 2. Comparison of demographic data between bots and humans' characters in B&S.


Fig. 3. Comparison of job distribution in Aion.

Most blocked users spent more than eight thousand minutes (130 hours) per week playing the game while a significant number of unblocked users spent only less than five thousand minutes (80 hours) (see Fig. 4-(a)). Even for a small number of unblocked users who spent similar amounts of time compared to blocked users, the frequency of activities conducted by unblocked users is significantly lower than that by blocked users (see Fig. 4-(b)). Consequently, the average activity frequency per unit time of bots is noticeably higher than that of normal users (see Fig. 4-(c)). That is, those characteristics might be useful to distinguish bots from human users. One interesting point is the statistics of each character's asset (see Fig. 4-(d)). Although blocked users play for longer time and carry out more activities, they tend to have fewer assets than unblocked users in a game. This is because they send their assets to other characters to be monetized as soon as there is a moderate accumulation of them, whereas normal users tend to accumulate wealth to buy expensive items. These statistics show that the ultimate goal of a bot is monetization, not the game enjoyment through the development of a character.

*3) Network analysis:* There are various social interactions in MMORPGs. Users can collaborate with other players to complete a quest, explore a dungeon or kill a giant monster by organizing a party. Besides, users can conduct economic activities by interacting with each other, *e.g.*, selling, buying, exchanging items, participating in an auction and so on. GFGs construct hierarchical systems for efficient economic activities. As a result, their social network structure characteristics might be significantly different from human users' network characteristics [16], [19] and the differences between those networks could be used to detect bots in GFGs.

We examine bot users' network characteristics compared with normal human users' network characteristics. We are particularly interested in their item trade networks where nodes represent users in a MMORPG and edges represent the trade relationships between users. Fig. 5-(a) shows the network structure of item trade between blocked users while Fig. 5-(b) shows the network structure of item trade between unblocked users. As shown in Fig. 5-(a), blocked users tend to trade items with a specific character (inner circle) that collects items from many characters (outer circle). This is because the aim of blocked users is usually to construct an efficient system via economic specialization. Unlike blocked users, unblocked users tend to trade with a few users; consequently, their network structure is moderately decentralized (see Fig. 5-(b)).

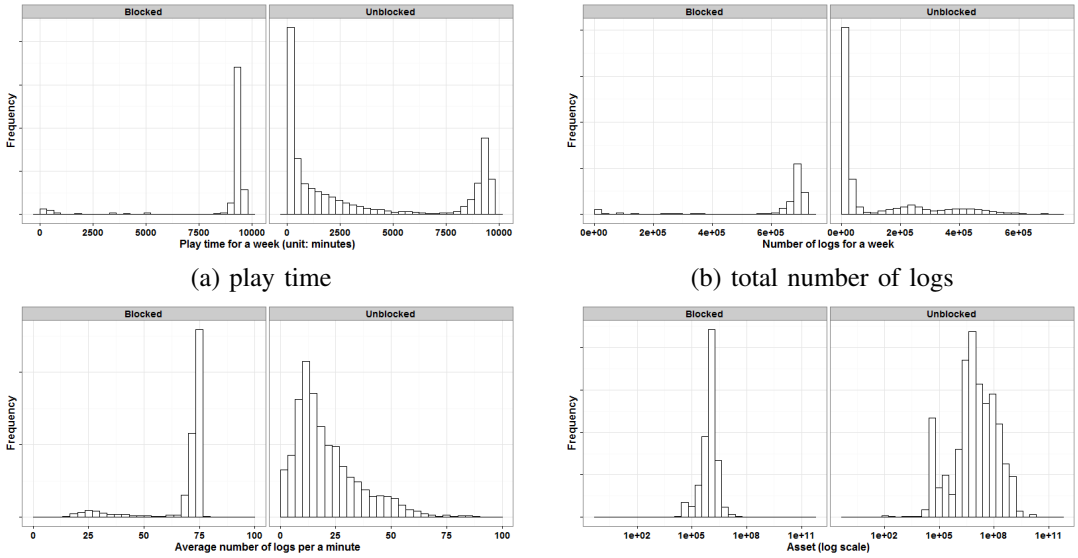
*4) Sequence data:* In [20], sequence mining techniques were used to detect bots and it was found that there is a difference in sequence patterns between bots and human players. We also examine that same characteristics in this work.

Fig. 6 shows that blocked users generate a large quantity of sequences, while the types of sequences are less varied compared to the amount of data because bot characters tend to perform repetitive actions. Fig. 7 clearly shows this characteristic. We can divide characters into two groups according to the ratio of unique count to the total count of sequence data in Fig. 7.

Although sequence pattern is a clear feature for detecting bots, we tried to check whether this pattern appeared in a more coarse-grained unit of data than sequence data in order to reduce data processing cost (which would occur if the pattern is valid in coarse-grained data). Consequently, we grouped logs by period, transformed them into vectors instead of sequences, and then computed the cosine similarity between log vector and unit vector to measure repetitiveness. For users who carry out various activities, various vectors are generated and various cosine similarities are measured. Fig. 8 shows the time series plots of cosine similarity for bots and normal users. As can be seen, bots have cosine similarities with fewer variations than normal users; therefore, we can use log vectors instead of sequences. Furthermore, we can measure the self-similarity of each user using the cosine similarity. Fig. 9 shows that bots can be effectively separated from humans using self-similarity. § IV-A describes this process in details.

## B. Feature review

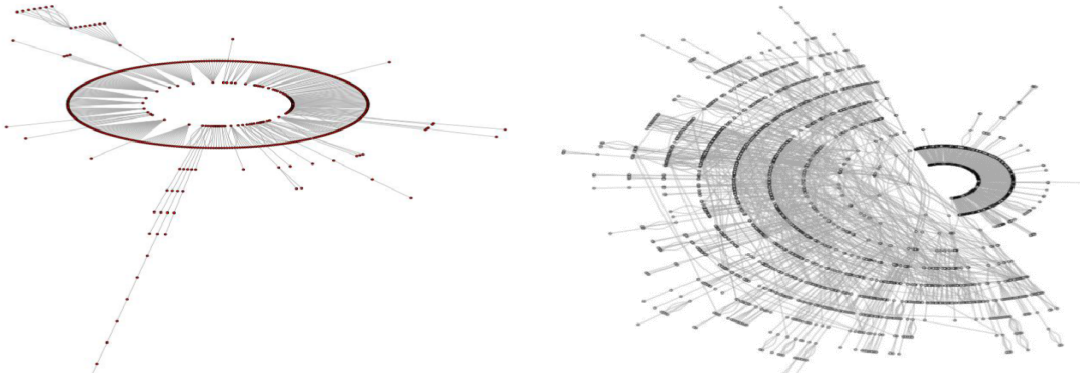
Through various forms of exploratory analysis in the previous section, we discovered several patterns for distinguishing




(a) play time (b) total number of logs

(c) average number of logs per unit time (d) asset

Fig. 4. Comparison of activity statistics between blocked and unblocked users: left box is blocked (bots), right is unblocked (normal users).



(a) blocked users (b) unblocked users

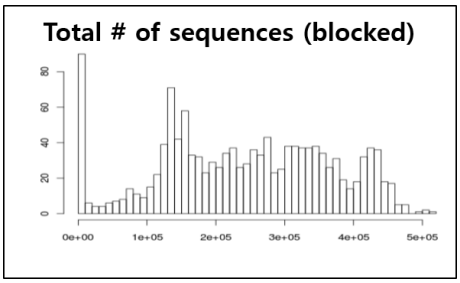
Fig. 5. Network structures of item trade: (a) trade with blocked users, (b) trade with unblocked users.

between bots and humans. To implement our detection system in the target games, we reviewed the results of data analysis and chose features using the following consideration.

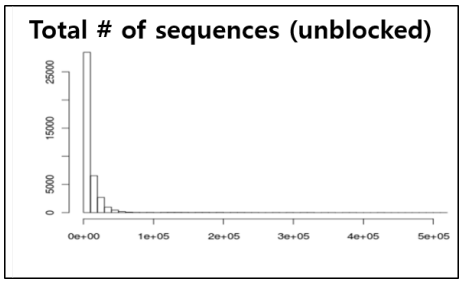
*1) Generality:* We wish to apply features to all target games. In Blade & Soul, bot users prefer a specific job type. However, Aion does not exhibit such bias for job preference. Therefore, the statistics of activity depending on contents in a specific game are not suitable. For example, the behavior pattern in a special dungeon (special dungeon is different in each game), count of quest completion (Lineage only has a few quests), and ratio of time required to make an item to the total play time (users need no time to make items in Blade & Soul) are not appropriate. On the other hand, bots tend to play longer and perform more actions than humans (see Fig. 4). Thus, we can use this pattern as a common feature for all target games. By the same reasoning, self-similarity is also a suitable discriminative feature (see Fig. 9).

*2) Stability:* An online game is updated periodically and continuously to fix bugs of client software or to maintain the service. Major updates force users to change their play patterns because game contents are added or the influence of attributes on game balance may be changed. Hence, bot programs are also updated or forced to change their behavior rules. For this reason, a demographic pattern is not stable because job preference would change if another job becomes an efficient job, *e.g.*, hunting of monsters. Statistics associated with economic values such as the amount of money a character trades with others or earnings per unit time is also not stable, because of inflation in the game world.

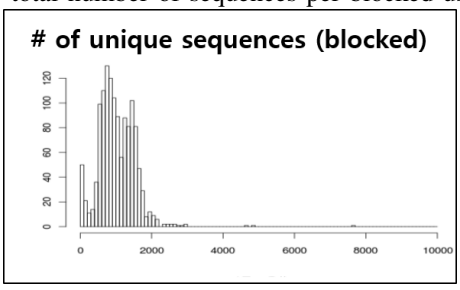
*3) On the difficulty of evasion:* In practice, simple threshold-based detection rules are straightforward and easy to implement. However, those rules can be circumvented by bots when threshold values are exposed. For example, if we use the threshold-based play time as a feature to detect bots, an attacker may play the game for less than the pre-determined threshold time in order to evade the detection rule.



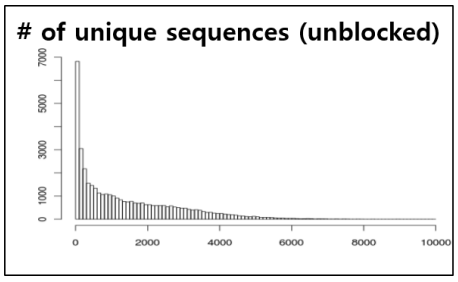

(a) total number of sequences per blocked user




(b) total number of sequences per unblocked user

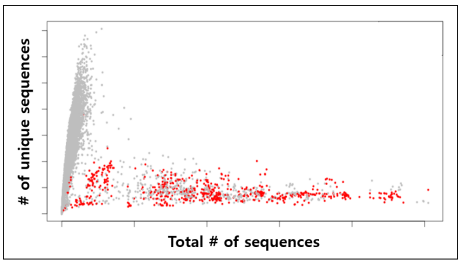



(c) number of unique sequences per blocked user

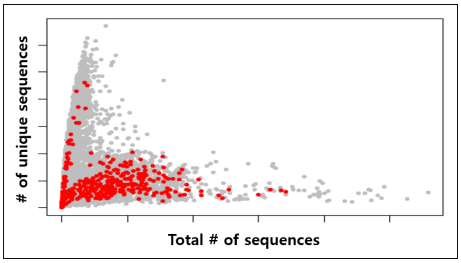



(d) number of unique sequences per unblocked user

Fig. 6. Histogram of sequence data.




(a) B&S




(b) Aion

Fig. 7. Scatter plot comparing number of unique sequences with total number of sequences per user. Almost blocked users (red) have lower ratio of number of unique sequences to total number of sequences than unblocked users (gray).

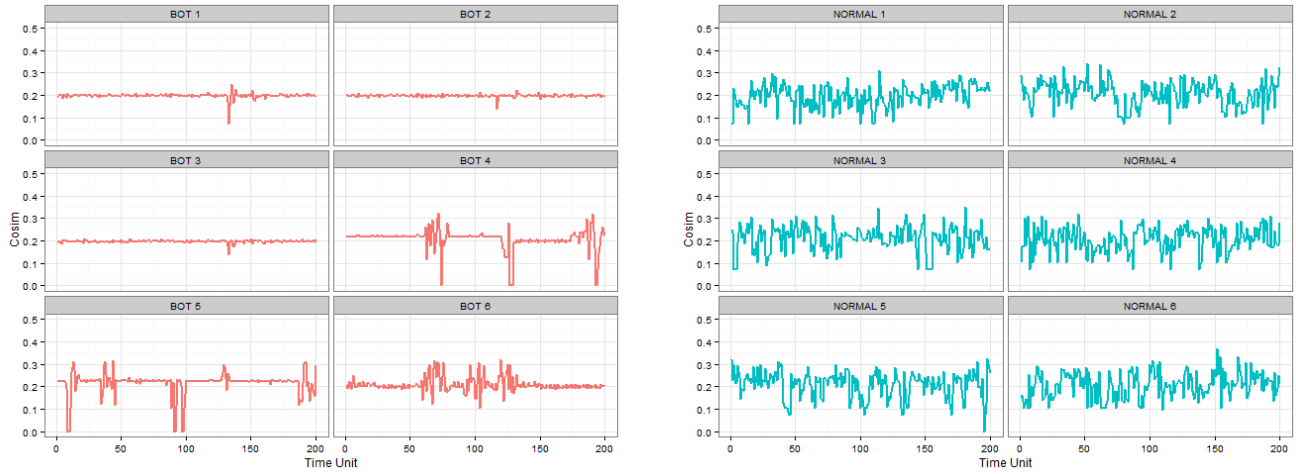
On the other hand, the proposed self-similarity detection system is more robust than the simple threshold-based bot detection system. Since our technique focuses on the frequency of repeated activities, rather than activities themselves, attackers should try their best to generate a new series of activities every time, which inherently makes their attacks more challenging and expensive (*i.e.*, slowing down the attack speed). For example, to lower the self-similarity, bots cannot simply repeat the same sequence of actions, which is a strategy in their best economical interest. Diverging from that strategy, however, would decrease the bots' overall productivity. When we analyze each user's self-similarity and the amount of game money that the user earned, there is a positive relationship between them (see Fig. 10). Even when a small portion of the total activities has random characteristics, the self-similarity measure is not significantly affected. This is because high-frequency activities are usually not altered, which dominate the self-similarity measure outcomes.

Moreover, unlike exiting game bot detection solutions using a small number of specific features, the proposed system is designed using all of the possible user data contained in game log files, which are automatically kept up-to-date with software updates and patches. Such property of our system makes it harder for the adversary to evade.

*4) Ease of development:* Ease of development is the important consideration because we wish to implement a practicable system for detecting bots. Features of the network structure can very effectively detect bots; however, we did not use them because they are difficult to compute in large data sets in our development environment for data processing (using Hadoop; iteration operations for graph algorithms cannot be simply implemented in Hadoop [23]). Conversely, self-similarity is highly appropriate for our system because of its simplicity. In particular, measuring self-similarity per user can be efficiently calculated with Hadoop in a parallel manner.

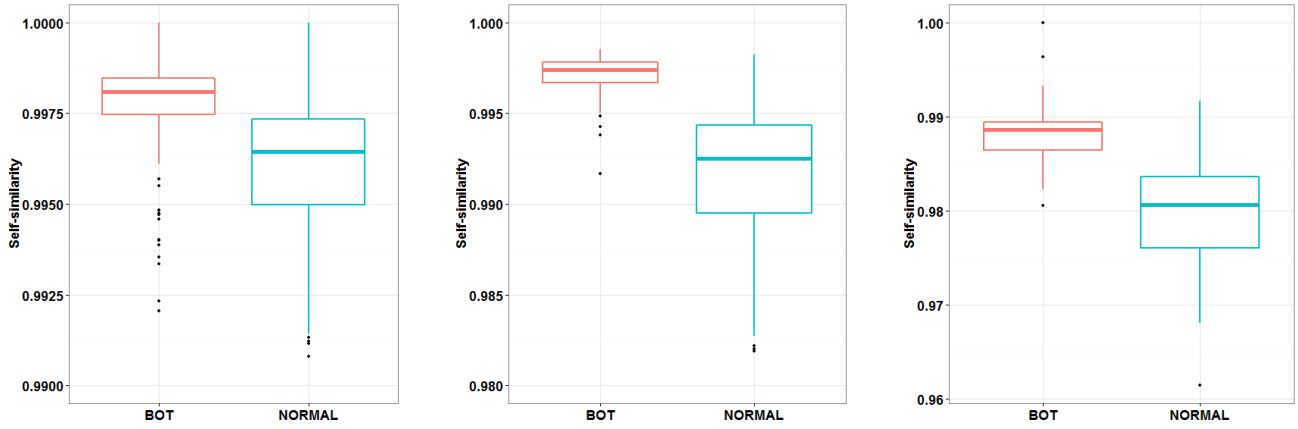
## IV. Methodology

We propose a systematic and integrated framework for game bot detection. The proposed framework comprises generic feature extraction, feature selection, modeling, evaluation, and an automatic tuning process that monitors the performance of the detection model and revises it autonomously when it detect changes. In §IV-A, we elaborate on the feature extraction and selection methods, especially the algorithm for measuring self-similarity of game users. §IV-B introduces the model learning and evaluation process. Finally, remaining




(a) bots (b) normal users

Fig. 8. Time series plot of cosine similarity.




(a) Lineage (b) Aion (c) B&S

Fig. 9. Comparison of self-similarity between bots and normal users.

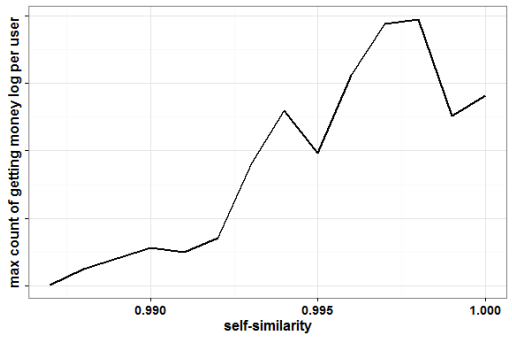



Fig. 10. Productivity in relation to self-similarity. X-axis is self-similarity, Y-axis is the maximum count of logs getting game money per user.

subsections explain the process of monitoring and maintaining the detection model.

### A. *Feature extraction and selection*

We introduce a self-similarity feature that measures the degree of repetitiveness of behaviors. The main feature of our model is a measurement of the self-similarity per game user, with the algorithms for its calculation detailed below.

*1) Generating log vectors:* We group game logs per user and sort them by time. We then set a time interval and transform logs in a time interval into vectors consisting of event id and its frequency per user. Fig. 11 describes the process of log transformation into vectors.

*2) Measuring the cosine similarity between log vectors:* We measure the cosine similarity between each log vector and the unit vector in which all elements are one. The cosine similarity between two vectors is defined as follows:

$$cos(\theta) = \frac{AB}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2}\sqrt{\sum_{i=1}^{n}(B_i)^2}}. \quad (1)$$

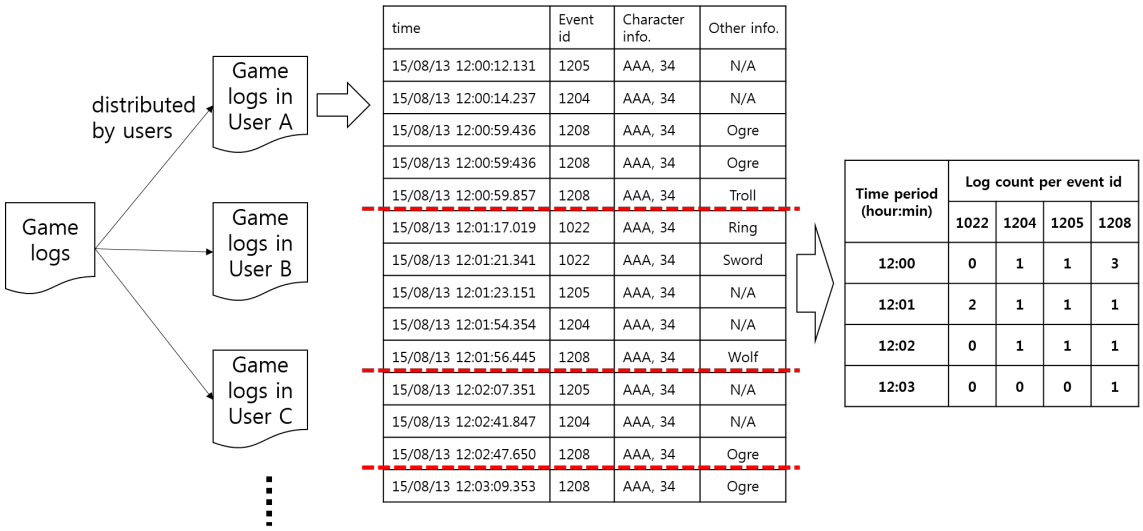



Fig. 11. Process for transforming game logs to vectors. In the above, four vectors are generated: $(0, 1, 1, 3), (2, 1, 1, 1), (0, 1, 1, 1), (0, 0, 0, 1)$.

TABLE II. LIST OF FEATURES USED IN THE PROPOSED DETECTION METHOD.

| No. | Feature | Description |
|---|---|---|
| 1 | self sim. | self-similarity |
| 2 | vector count | count of a set of log vectors |
| 3 | uniq. vector count | unique count of a set of log vectors |
| 4 | cosim. zero count | count of data in which cosine similarity is zero in a set of log vectors |
| 5 | vector mode | count of data that appears most often in a set of log vectors |
| 6 | total log count | total count of logs generated by user |
| 7 | char. level | character level |
| 8 | play time | play time during certain period per user |
| 9 | npc kill count | NPC kill count |
| 10 | trade take count | count of trade in which user takes items from another |
| 11 | trade give count | count of trade in which user gives items to another |
| 12 | retrieve count | count of activity in which user retrieves items from warehouse |
| 13 | deposit count | count of activity in which user deposits items to warehouse |
| 14 | log count per min. | average count of logs are generated per minute |

*3) Measuring self-similarity:* The logs of each user are transformed into sets of cosine similarities following the above steps. To measure self-similarity, we calculate the standard deviation of the cosine similarities and then transform it using Eq. (2). The transformed value is close to one when self-similarity is high and close to 0.5 otherwise, which is the same scale as the Hurst exponent [3]:

$$H = 1 - \frac{1}{2}\sigma, \tag{2}$$

where $\sigma$ is the standard deviation of the cosine similarities and $H$ is the self-similarity.

*4) Additional feature selection:* Our proposed self-similarity algorithm has one shortcoming. If a character only plays for a very short period or performs no activities over a long period of time, self-similarity for the character may have a high value. Because that is not our intention, we added several features to address this shortcoming. When we select complementary features, we consider the attributes offered in general MMORPGs. Table II lists all the features we used.

## B. Modeling and evaluation

We used logistic regression to detect game bots. Our model calculates the probability of a character being a game bot by using the features in Table II as independent variables. We evaluated the performance of our detection model using $k$-fold cross validation. In $k$-fold cross-validation, ground truth is partitioned into $k$ subsets of the same size. A single partition is used as the validation for the model, and the remaining partitions are used for training. After the previous step, another partition is selected as a test set (among the $k-1$) while others are used as the training set. After cross-validation is repeated $k$ times, the performance measure per partition is averaged. We used the area under the Receiver Operating Characteristic (ROC) curve to evaluate the performance of our model. The ROC curve is one of the methods used to measure the performance of binary classifiers. This is a graph that shows the false positive rate (FPR) on the x-axis and true positive rate (TPR) on the y-axis according to a threshold. TPR and FPR are defined as follows:

$$\text{FPR} = \frac{FP}{FP+TN} \text{ and } \text{TPR} = \frac{TP}{TP+FN}, \tag{3}$$

where $FP$, $FN$, $TP$, and $TN$ are the false positive, false negative, true positive, and true negative, respectively. In general, if the threshold is higher, *i.e.*, the standard for detecting game bot is stricter, FPR and TPR become lower, and vice versa. The performance of the model is deemed to be good when

the false positive rate is low, while true positive rate increases when there is a low threshold. Accordingly, the ROC curve is created as a curved line to the left and upper side, so the area under curve increases.

### C. Monitoring and retraining detection model

In this step, we propose a detection model maintenance algorithm. The algorithm detects the change in performance of the underlying model and retrains the model automatically when the game developer updates the game contents or game bots change their behavior. Our system calculates the game bot probability of all users periodically. If the coefficient of probability between two periods changes radically, the system retrains and validates the suitability of the model. We implemented the system to notify operators of the results of retraining and validation. Fig. 12 outlines the maintenance process of our system. In the process, we use the Exponential Weighted Moving Average (EWMA) method, which was originally developed for quality management [12], to detect changes in the model performance. The details of the algorithm using EWMA are given below.

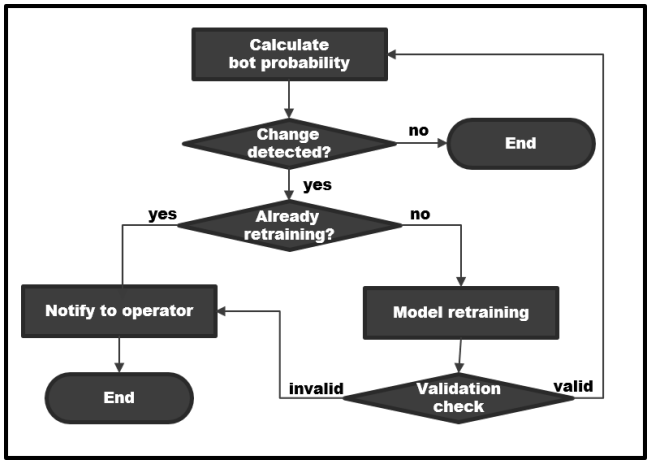



Fig. 12. Automated maintenance process.

*1) Calculating the coefficient:* The coefficient of game bot probability between time $t$ and $t-1$ is calculated as:

$$x_t = \frac{n\sum a_i b_i - \sum a_i \sum b_i}{\sqrt{n\sum a_i^2 - (\sum a_i)^2}\sqrt{n\sum b_i^2 - (\sum b_i)^2}}. \quad (4)$$

In Eq. (4), $a_i$ signifies the probability of user $i$ being a game bot at time $t$ and $b_i$ signifies probability of user $i$ being a game bot at time $t-1$.

*2) Calculating variance of coefficient:* We calculated the weighted moving average in the coefficient measured in the previous step. In this equation, $\lambda$ is the weighted value.

$$z_t = \lambda x_t + (1-\lambda) z_{t-1}, 0 < \lambda <= 1, t = 1, 2, ... \quad (5)$$

*3) Measuring upper and lower control limits:* In this step, we define the upper and lower control limits for evaluating a change in the performance of the model. If $z_t$ in Eq. (5) is out of the control limit, we consider that the behavior pattern of the game bots has changed. The control limit has upper and lower lines as defined in Eq. (6).

$$CL = \mu \pm L\sigma\sqrt{\frac{\lambda}{z-\lambda}}. \quad (6)$$

In Eq. (6), $\mu$ is the average of $z$ and $\sigma$ is the standard deviation of $z$. $L$ is a constant that determines the sensitivity of the pattern's change. If $L$ is small, the algorithm detects small changes and notifies operators. Otherwise, the situation is considered as a normal state.

*4) Retraining model validation:* If $z$ is beyond the control limit, the model is retrained autonomously. The retrained model is then validated before reapplying it to bot detection in the system. Area Under Curve (AUC) is used for the validation check through $k$-fold cross validation. If AUC is lower than 0.9, the system judges to be invalidated and notifies operators.

## V. Experiments

### A. Data preprocessing

We implemented the data preprocessing modules to extract features in three games: Lineage, Aion, and Blade & Soul. The total number of log types was in the range of 200 to 300 per game. However, we only selected logs directly associated with characters' activities to measure self-similarity accurately. The size and volume of data in each game are summarized in Table III. The total numbers of logs and users were aggregated over one week.

TABLE III. SUMMARY OF RAW DATA (B:BILLIONS, K:THOUSANDS).

| Game | Logs | Types of logs | Users |
|---|---|---|---|
| Lineage | 28.5B | 165 | 570K |
| Aion | 5.0B | 229 | 250K |
| B&S | 6.9B | 109 | 130K |

We also determined two important parameters for processing data: the period for aggregating log data, and the time unit of log vectors for measuring self-similarity. Our target games have down time every Wednesday morning because operators need to check the servers and update contents. Consequently, most users have a one-week playing cycle. As a result, we decided to aggregate feature data over one week. If the time unit of a log vector is too short, the amount of information in the vector is too small. However, the converse results in the number of vectors being so small that self-similarity cannot be measured accurately. We tested the detection model using only self-similarity for a feature in the range of 300–1800 seconds and checked the AUC of each model to determine the optimal time unit for a vector. Fig. 13 describes the results of our test.

In Fig. 13, the x-axis is a period of aggregation, and the y-axis is the AUC of models using the $k$-fold cross-validation. Each plot line depicts the AUC of a model per time unit that we tested. In our experiments, the shorter time unit of a vector is better. A five-day aggregation period is best. However, we selected seven-day as a parameter because it is

more convenient and efficient to synchronize with the server maintenance time. In addition, the difference in AUC between five days and seven days is substantially small for the time unit of 300 seconds. We decided on 300 seconds for the time unit of a vector. We did not conduct any more tuning for the time unit of the vector less than 300 seconds because as the time unit decreases, the amount of system resources needed increases, although the model performance could be better.

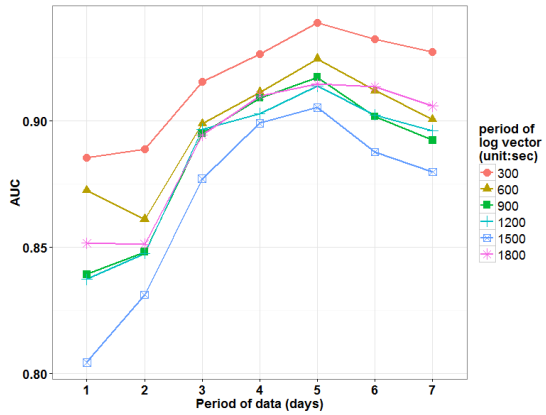



Fig. 13. AUC of detection model per vector period.

### B. Ground truth construction

Upon establishing the game bot detection model, we constructed the ground truth for its learning and evaluation. In previous studies, some researchers used data sets generated artificially via several human and bot programs [24]. Other researchers used a list of banned accounts that operators caught, or other players reported [1]. However, these approaches may generate false negatives. Ahmad *et al.* [22] proposed a capture-recapture technique and network analysis for resolving this problem. However their method can still be inaccurate because of the absence of reliable ground truth.

In this work, we constructed a reliable ground truth by sampling and coding and through thorough visual inspection (through a time-consuming process). In practice, the game company has game masters who monitor the game world, detect malicious users, and deal with users' arguments and complaints. They use special characters, called the Game Master (GM) character. A GM character has extra abilities not available to a normal character. One such ability is "teleportation", which allows the GM character to instantly move to the zone where a character of interest is playing at that time. Another ability is "invisibility", which allows them to observe other characters play without them being seen or noticed. For our ground truth, we relied on those GMs. For that, they carefully observed samples randomly selected and coded to determine whether an individual is a game bot or not. We provided precise instruction rules and process to avoid error due to the GMs' subjective bias or mistakes. The process is detailed in the following.

First, as a baseline, we required that at least two GMs observe a character at the same time and independently judge whether the character was a game bot or not. We collected the evaluation results for all samples and only used those inspection results that were in total agreement. Second, we asked the GMs to describe the reason for his or her judgment. After collecting the results, we arranged and standardized the reasons. Third, we used the Leave One Out Cross Validation (LOOCV) method to code the results for cross-checking. LOOCV is a technique in which a sample is used for testing while all others are used for training the model. When the coder leaves a wrong code for the sample, we can identify the incorrectly coded sample using LOOCV. The first two rules may reduce the problem of subjective bias, and the last rule may prevent human mistakes from influencing the final selection of ground truth.

TABLE IV. GROUND TRUTH CONSTRUCTION RESULTS.

| Game | Bot | Human | # of inspector | $\kappa$ |
|---|---|---|---|---|
| Lineage | 315 | 244 | 3 | 1.0 |
| Aion | 75 | 75 | 2 | 1.0 |
| B&S | 260 | 230 | 3 | 1.0 |
| Total | 650 | 549 | - | - |

We built ground truth data sets for Lineage, Aion, and B&S. Our ground truth was gradually built over a period of time for each game product from December 2013 to March 2015. The total ground truth list is summarized in Table IV.

### C. Modeling and evaluation

We used logistic regression to detect game bots. Each feature was estimated using the regression coefficient of the regression model, and then each coefficient was applied to Eq. (7) to calculate the game bot probability.

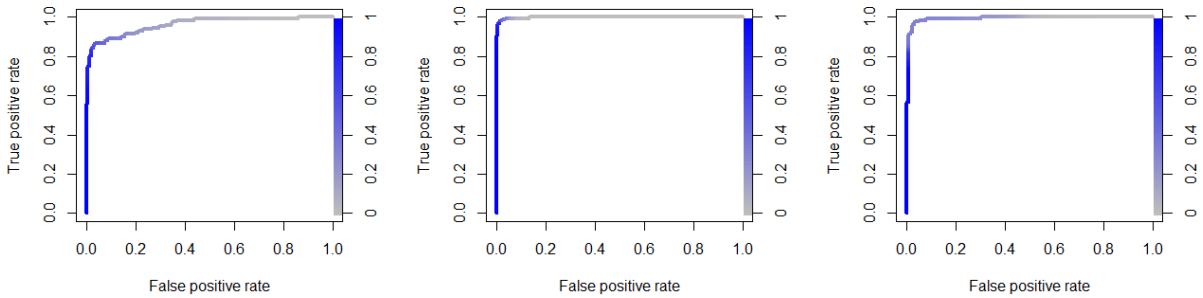
$$P_{BOT} = \frac{1}{1 + e^{-(\alpha + \sum_{i=1}^{n} \beta_i x_i)}}, \tag{7}$$

where $\alpha$ and $\beta$ are intercept and coefficients, which are calculated by logistic regression, respectively.

The result of logistic regression for each game is summarized in Table VI. The statistically significant feature set applied to regression was different from all others in each game because game contents and characteristics are determined per individual game. However self-similarity is used as a common feature and has statistical significance in all games.

We used $k$-fold cross-validation to evaluate the performance of the detection model. We set $k$ to 10; thus, the AUC of our model was measured 10 times. Consequently, we used the average of the set of AUC as a performance measure. We constructed the detection system with the self-similarity alone and then tried to optimize the system with some additional features (mentioned in §IV-A-4)) The final results of the evaluation are shown in Table V and depicted in Fig. 14.

The size of the sample sets of players used for evaluation in Table V and the size of the ground truth sample set in Table IV are different. Our ground truth for each game was gradually constructed over several months, not at once. Consequently, we could not use the entire set of the ground truth when we evaluate the model because some users did not play the game at the time of the evaluation. In the case of Aion, the ground truth we built was small, so we boosted the data by processing game logs for users who played the Aion in several periods.




(a) Lineage (b) Aion (c) B&S

Fig. 14. ROC curve of detection model.

TABLE V. PERFORMANCE OF DETECTION MODEL.

| Game | Bot | Human | AUC (initial model) | AUC (final model) |
|---|---|---|---|---|
| Lineage | 128 | 149 | 0.8967 | 0.9455 |
| Aion | 186 | 160 | 0.9557 | 0.9942 |
| B&S | 131 | 129 | 0.8280 | 0.9399 |

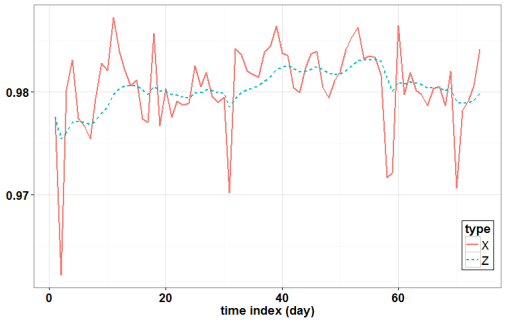



Fig. 15. Variation of $X$ and $Z$ values by date.

### D. Automated model maintenance

We implemented a system that calculates the game bot probability of users and regularly tracks the number of game bots in a game world. We believe the proposed detection model is sufficiently reliable. However, the playing pattern of a game bot changes for a variety of reasons. For instance, game contents are updated, bot programs get updated, and bot users change the setting of the bot program to evade detection. The detection model can become invalid in such cases because the regression coefficient of the detection model can be unfit for new users. For this reason, the detection model should be revised by retraining with the newest ground truth data when the model becomes invalid. A system that detects such changes and automatically retrains the detection model at that time is required to do this.

The flow of the overall process is depicted in Fig. 12. The monitoring algorithm, *i.e.*, EWMA, for model validation is explained in §IV-C. To implement the monitoring algorithm, we need to define several parameters; specifically, $\lambda$, the weighted constant for moving average in Eq. (5); $L$, the constant for control limit and $n$, the window size for the average and standard deviation of $Z$'s in Eq. (6).

TABLE VI. REGRESSION COEFFICIENT OF FEATURES.

| variable | regression coeff. | std. error | z value | p value |
|---|---|---|---|---|
| **self sim.** | **-3.757e+00** | **1.399e+00** | **-2.686** | **0.00724 **** |
| cosim. count | -1.439e-02 | 2.006e-02 | -0.717 | 0.47325 |
| cosim. uniq. count | -1.716e-03 | 3.796e-03 | -0.452 | 0.65132 |
| cosim. zero count | -6.139e-03 | 5.625e-03 | -1.091 | 0.27516 |
| cosim. mode | -8.852e-03 | 8.966e-03 | -0.987 | 0.32348 |
| total log count | -9.493e-06 | 6.379e-06 | -1.488 | 0.13672 |
| **main char. level** | **-8.391e-02** | **2.700e-02** | **-3.108** | **0.00188 **** |
| total use time min. | 4.377e-03 | 4.119e-03 | 1.063 | 0.28798 |
| npc kill count | -1.176e-05 | 6.964e-05 | -0.169 | 0.86585 |
| trade get count | -4.827e-02 | 4.298e-02 | -1.123 | 0.26133 |
| trade give count | 1.056e-02 | 4.268e-02 | 0.247 | 0.80461 |
| **retrieve count** | **1.582e-02** | **5.266e-03** | **3.003** | **0.00267 **** |
| **deposit count** | **-1.040e-02** | **4.779e-03** | **-2.175** | **0.02961 *** |
| log count per min. | 1.808e-02 | 1.547e-02 | 1.168 | 0.24267 |

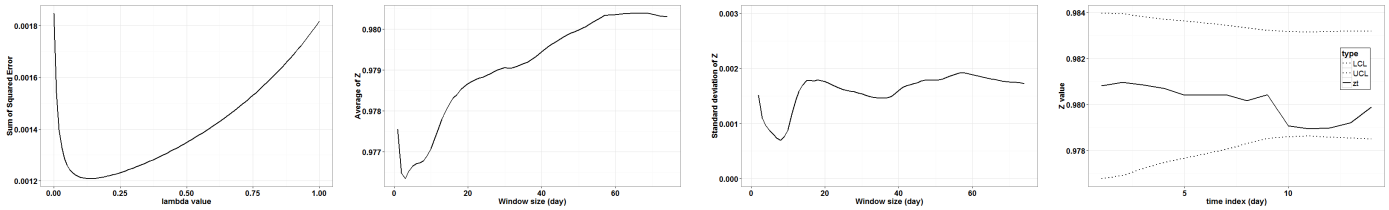
(a) Lineage

| variable | regression coeff. | std. error | z value | p value |
|---|---|---|---|---|
| **self sim.** | **1.337e+03** | **5.041e+02** | **2.651** | **0.00802 **** |
| cosim. count | 6.678e-02 | 1.470e-01 | 0.454 | 0.64966 |
| cosim. uniq. count | -2.378e-02 | 6.557e-02 | -0.363 | 0.71686 |
| cosim. mode | -2.444e-01 | 1.464e-01 | -1.670 | 0.09494 |
| total log count | -8.542e-05 | 1.280e-04 | -0.667 | 0.50470 |
| **main char. level** | **-1.809e-01** | **8.013e-02** | **-2.257** | **0.02399 *** |
| total use time min. | -1.550e-04 | 6.382e-03 | -0.024 | 0.98062 |
| **npc kill count** | **-2.551e-04** | **1.002e-04** | **-2.546** | **0.01091 *** |
| trade get count | 3.648e-03 | 1.522e-02 | 0.240 | 0.81056 |
| trade give count | -8.576e-03 | 3.674e-02 | -0.233 | 0.81541 |
| log count per min. | -1.371e+00 | 1.028e+00 | -1.334 | 0.18235 |

(b) Aion

| variable | regression coeff. | std. error | z value | p value |
|---|---|---|---|---|
| **self sim.** | **-6.469e+00** | **2.874e+00** | **-2.251** | **0.02439 *** |
| cosim. count | -3.923e-02 | 5.063e-01 | -0.077 | 0.93824 |
| cosim. uniq. count | -4.157e-02 | 5.051e-01 | -0.082 | 0.93442 |
| cosim. zero count | -6.110e-01 | 3.413e-01 | -1.790 | 0.07345 |
| cosim. mode | 7.520e-01 | 6.426e-01 | 1.170 | 0.24190 |
| **total log count** | **1.026e-04** | **3.265e-05** | **3.143** | **0.00167 **** |
| main char. level | 1.002e+00 | 2.661e+01 | 0.038 | 0.96995 |
| total use time min. | 3.965e-05 | 7.415e-04 | 0.053 | 0.95735 |
| npc kill count | 1.540e-05 | 1.538e-04 | 0.100 | 0.92023 |
| trade get count | -7.175e-04 | 1.525e-03 | -0.471 | 0.63796 |
| trade give count | -3.292e-04 | 9.725e-03 | -0.034 | 0.97300 |
| log count per min. | -2.050e-02 | 2.671e-02 | -0.768 | 0.44278 |

(c) B&S




(a) variations of SSE by $\lambda$ (b) variations of $Z$'s average by window size (c) variations of $Z$'s std. deviation by window size (d) variations of $Z$ and control limit by date

Fig. 16. EWMA parameter tuning.

$Z$ is the weighted moving average of $X$ (see Fig. 15) and is used to estimate correlation coefficient, *i.e.*, $X$ at time $t+1$ in this algorithm. Consequently, we selected the $\lambda$ value so that $\lambda$ minimizes the Sum of Squared Error (SSE) between $X_t$ and $Z_{t-1}$. In Fig. 16-(a), SSE decreases at the outset, but increases after a certain point. Therefore, the inflection point is the most suitable value for $\lambda$, which we used.

For window size $n$, we observed the average and standard deviations of $Z_t$, $Z_{t-1}$,$\cdots$, $Z_{t-(n-1)}$ by n changing incrementally as shown in Figs. 16-(b) and (c). As $n$ gets larger, indicating that the control limit considers the longer period, the average and standard deviation increase and converge eventually. We aimed to design the robust control limit, so we determined $n$ where the average and standard deviation converge. Based on experiments, we found that $n = 60$ is a reasonable value to use.

Finally, we optimized the control limit constant $L$. For that, we observed the changes in $Z$ for optimal $\lambda$ and $n$ (see Fig. 16-(d)). We could fix $L$ at a minimum integer value under the maximum variance of $Z$.

Table VII describes optimal values for the parameters for EWMA of the target games.

TABLE VII. EWMA PARAMETERS.

| Game | $\lambda$ | $n$ | $L$ |
|---|---|---|---|
| Lineage | 0.15 | 60 | 20 |
| Aion | 0.95 | 60 | 4 |
| B&S | 0.15 | 60 | 38 |

### E. Ethical considerations

Before installation of a game client program, participants were asked to acknowledge a consent form under the End User License Agreement (EULA) and domestic laws, which informed them that their data may be used in improving the quality of the installed game. Anonymous data samples were collected confidentially only for the purposes of conducting statistical analyses. Ethical compliance of our research was also validated through an ethics committee at our universities (IRB and equivalents).

## VI. REAL-WORLD DEPLOYMENT

We have implemented and ran this proposed detection system in multiple commercial MMORPGs for more than two years, from Feb. 2013 to date. As a result, we were able to detect and block more than 15,000 bot users. We note that a part of bot users were just banned for business decisions.

Fig. 17 is the screenshot of our bot detection system to monitor three large commercial MMORPGs (about 280,000 concurrent users). It automatically detects bot players and gives useful statistics (bot growth trend, in-game currency, *etc.*).

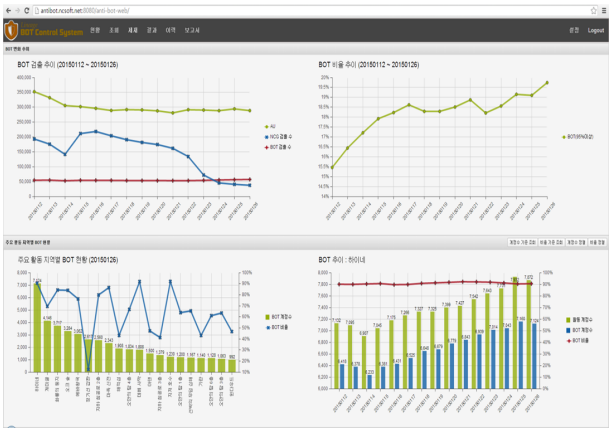


Fig. 17. Screenshot of the bot detection system.

In the following, we provide a postmortem report based on our two year's detecting and banning operation.

**Generality.** To implement this system, we had one year of observational period for each game. After we implemented the detection system for Lineage, we expanded our work to other games. When we applied our techniques to Lineage, we needed one year to develop the system and tune its parameters. With the gained experience, the implementation of the detection system for Aion and Blade & Soul took only two months. Main algorithms are reused and only a few game specific features are updated for each implementation case. Such reusability shows that our proposed system can be widely applied to the same genre of MMORPGs.

**Ground truth and model maintenance.** For model maintenance, we try to generate new ground truth data steadily. Whenever our detection model is not working well (*e.g.*, due to the update of game contents), we need to retrain the detection model with the ground truth data. This process can automatically be performed without substantial user intervention. In addition, we realized that inspecting task for creating ground truth does not have to be performed by highly-skilled GMs if the inspection process that is described in §V-B is well operated. Consequently, we can reduce the cost of establishing a new ground truth.

**GFG economy.** Based on our deployment, we can estimate GFG's economic scale. For example, in the case of Aion, the average GFG has 83 bot users. We summarize the data of the detected bots that belong to GFGs in Table VIII. This table shows GFGs in a sample Aion server. For this analysis, our observational period was one month. First, the findings show that the size of the various GFGs varies, and ranges between six and 590 characters (with an average of 83 characters), thus highlighting a variety of actors and business sizes in this ecosystem. Furthermore, findings show that assets of smaller GFGs are non-negligible; for example, the smallest GFGs can earn more than $12,000$ USD worth of assets, while the largest earns more than $107,000$ USD in a month.

TABLE VIII. GFGs AND THEIR MONEY POSSESSION.

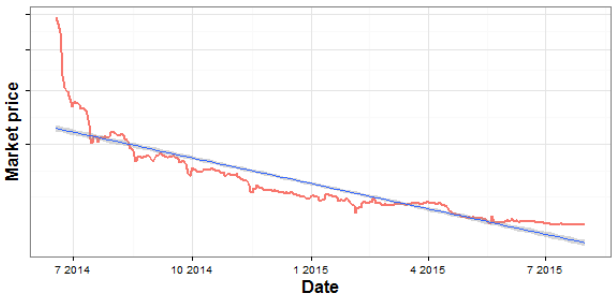
| | Nodes | Edges | Game Money | Real Money Value |
|---|---|---|---|---|
| Mean | 83 | 295 | 20,977,234,225 | $41,954 |
| Std. dev | 95 | 316 | 12,044,804,616 | $24,090 |
| Max | 590 | 1,964 | 53,699,776,093 | $107,400 |
| Min | 6 | 13 | 6,171,466,805 | $12,343 |

**Extreme cases and anomalies.** As we mentioned earlier, the self-similarity algorithm cannot detect bot users who play for a short period, which is a shortcoming of our system. This shortcoming is not a problem, in general, where the more common cases of bots have their models of incentives, where (almost always) bots' play time is longer than humans' (see Fig. 4-(a)). This behavioral feature shows that our criteria for distinguishing between bots and humans using similarity, in general, is reasonable to detect most bots.

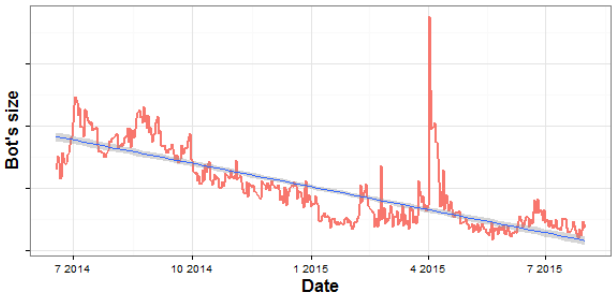
However, one operational incident that goes against the above criteria is worth mentioning, highlighting the shortcoming. In our evaluation period of our system, we observed a massive number of bots utilizing a single game server. Those bots were particularly playing for less than 10 hours per week (on average), while the average bots play time was about 160 hours for that week. In general, such anomaly (with respect to the expected behavior of bots) would make them undetectable. However, we believe that those bot behaviors are not common in practice. When bots run for less than 10 hours per week, they may not produce sufficient cyber assets for their benefits. For example, the subscription fee of Lineage is 25 USD per month, which is a value of approximately 6 millions gold units. If we assume that a bot can earn 50,000 gold units per hour, GFGs have to run their bots at least 120 hours per month (i.e., about 30 hours per week) to get some benefits. In fact, our assumption is reasonable because the maximum amount earned by a bot is less than 50,000 gold units in our data sets.

The only exception is the case when a new game is released or a new server is added to mature service. In this case, a market price of RMT is very high. For example, a market price of the new server in which short time playing bots were found was at least ten times higher than in other mature servers. Therefore, it seems enough for GFGs to run their bots for only a few hours to collect those items. Surely, this phenomenon diminishes as the economy grows, and the market value goes down over time (see Fig. 18).

Although this issue was an isolated case of anomaly, we believe that we need to complement our system to capture and detect such case early. We believe that network analysis




(a) market price of real money trading in new server




(b) short time playing bot's size

Fig. 18. The dynamics of the market price against the number of bots that play for a fewer number of hours.

is a strong complement to the methods we provided in this work. As Fig. 19 shows, short time playing bots tend to have a star-shaped trading network structure. Such structure as a feature could be used to detect those anomalies earlier in evaluation. The description of the detailed process of such combined process is left as a future work.

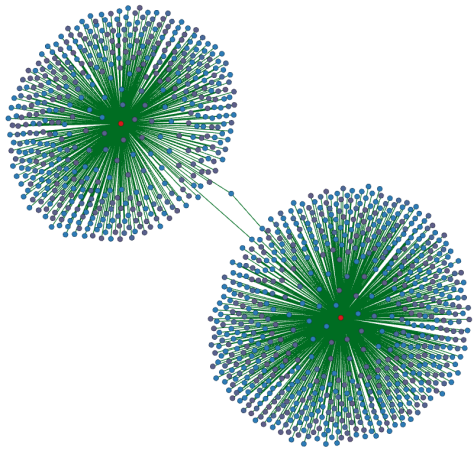


Fig. 19. Trading network of short time playing bots where nodes represent bot players and edges represent trading transactions between bot players.

**Hacks detection.** Our system does not address the 'hacks detection' issue. Although game hacks are one of serious security problems, they are not as prevalent as bots. In a nutshell, hacks are malicious software which make a character strong, fast or immortal via techniques like RAM or network traffic modification. Hacks have various functions to enable users

to possess game items and game money without purchase, speed up the game play, see through the wall, teleport illegally, give a fake report as a winner to the game company and so on. For this case, the network-side detection is a primary solution rather than the server-side detection that analyzes the user log. The network-side detection can identify network traffic burstiness and anomalies in the timing of commands and activities, response time, and traffic interval and timing channels by looking into packet-level network artifacts. For this type of cheating, we need a different model that focuses on understanding and investigating the network traffic and artifacts as a feature. Further, the detection model based on log analysis should be devised to identify the intermittent anomaly of users behaviors comparing their usual behaviors as well as others' behaviors. We believe that this issue is out of the scope of this work, although a worthwhile future work.

**False positive issue.** False positive should be avoided for the system to be practical. Banning innocent players causes users' churn, and may raise legal issues and concerns. To reduce false positives, we include additional filter to our system. While we mainly rely on in-game features (*i.e.* repetitive action, user's game playing behavior, *etc*.) in our model, we found that out of game information are very helpful in reducing false positives and the cost of these. For example, we can filter just bot users who have IP address that are used by other bots, or who have low business profile (*e.g.* payment's history).

**Individuals using bots.** Some individuals operate bots. The certain portion of bot usage is distributed among individual players. A human player is playing for several hours a day and then turns on a bot to farm for a few hours. Because these users show legitimate and fraudulent behaviors at the same time, and given that we consider the short aggregation period (*e.g.* twelve hours) for self-similarity, the self-similarity may go beyond the threshold value at times as shown in Fig. 20. However, in our system, and considering the 7 day aggregation period and the high threshold value, this type of users may evade our detection. In this study, we are only concerned with gold farming groups rather than individual bot players, and mainly because gold farming groups are more harmful to the system. However we believe that individual bot players also can be detected if we consider a variation of self-similarity for shorter period than shown Fig 20. We will study the approach for addressing those users in a future work.

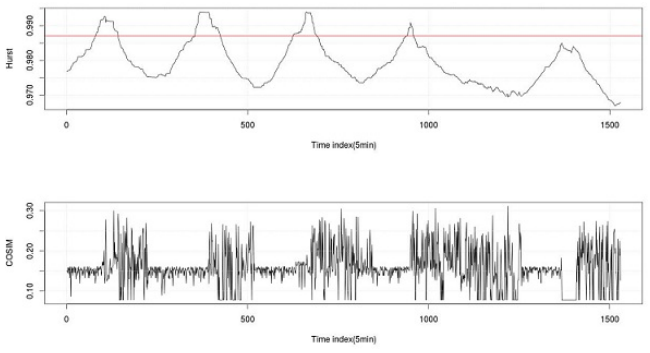



Fig. 20. Time series of self-similarity (upper) and cosine similarity (lower) of a human player who is suspected of using a game bot in part.

## VII. Related Work

Much research has been conducted on constructing game bot detection methods using machine learning techniques.

Ahmad *et al.* [1] conducted a pioneering study by evaluating the performance of various classification algorithms to detect bots in the game named "EverQuest II" (https://www.everquest2.com/). They coined the terms "gatherers", "bankers", "dealers", and "marketers" for categorizing bots with several features such as demographic data, characters' sequential activities, statistical properties, virtual item transactions and network centrality measures. However, their chosen features were not well developed and the used ground truth was not sufficiently reliable. Consequently, their detection performance was lower than our expectations.

Thawonmas *et al.* [24] also conducted an early study that tried to detect game bots using bots' inherent behaviors such as repeating the same activities more often than normal human users. Their proposed detection rules, however, were based on simple threshold values that cannot be easily determined in practice.

Bethea *et al.* [2] presented a defence technique using symbolic execution to analyze the client output and determine whether that output could have been produced by a valid game client. The effectiveness of the technique was verified through two case studies of customized games. However, their proposed technique cannot detect cheats that are permitted by the sanctioned client software due to bugs or modified game clients that do not change their behaviors as seen at the server.

Inherently, the performance of classification algorithms can be dynamically changed with the selected features. Unsurprisingly, using general features have relatively low performance compared with the classification algorithms with features specific to games and/or genre. Kang *et al.* [14] proposed a bot detection model based on the difference in chatting patterns between bots and human users. Kang *et al.* [15] also proposed a game bot detection method based on the players' network characteristics. Those proposed methods can detect bots with a high accuracy. However, the costs for building detection rules are expensive and an additional tuning process is needed because the methods generally depend on game contents and are sensitive to the configuration settings of bot programs.

In addition to individual game bot detection, researchers have also tried applying several different techniques to identify bot groups. Chung *et al.* [8] proposed a method that groups characters by behavioral similarities using the K-means clustering algorithm and then detecting bots in each group using Support Vector Machine (SVM). However, it may incur high costs to build a detection model and update it with dynamic changes in users' playing patterns. Mitterhofer *et al.* [21], Chen *et al.* [7] and Kesteren *et al.* [25] proposed similar methods with general features such as the moving paths of characters, respectively. Those proposals use the characteristic tendency of bots to move with fixed routes set up by bot programs. Unlike the classification algorithms using specific features, those methods can be applied to most MMORPGs. In their approaches, however, the log resolution of moving paths has to be increased in order to obtain a high accuracy. Consequently, the system has a large burden to process logs with a high resolution. In addition, it is easy to avoid detection by using

randomized moving paths which can be easily controlled by bot programs.

In this paper, we extend their works into a more generalized model; while their approaches [7], [25], [21] simply used the single feature of moving path, we build a generalized framework with various features by designing a self-similarity algorithm to effectively measure bots' repeated activity patterns, which was previously used as a means of analyzing network traffic [10] or developing intrusion detection systems [18]. We note that the proposed method is significantly robust to changes in the configuration settings of bot programs compared with existing approaches (e.g. [7], [25], [21]) because our method focuses on all behaviors and long-term activities.

## VIII. Conclusion

In this paper, we introduced the first study on deploying an automated model for detecting game bots in real MMORPG services. Our evaluation results show that a practically effective classification algorithm can be constructed by identifying game bots' repeated activity patterns which are clearly distinguished from normal human users' patterns. We validated the effectiveness of the proposed framework by evaluating the performance of the classification algorithm with ground-truth data and deploying it on real MMORPGs ("`Lineage`", "`Aion`" and "`Blade & Soul`"). Although, we currently limited our experiments in those three MMORPGs, we believe that the proposed detection system can also be an efficient tool to fight game bots in other MMORPGs. The analysis and ground-truth collection are naturally facilitated by the operation of the three studied MMORPGs, and the system can be easily extended to other games. In the future, and along with extending the work to other games, we will explore using network features for more robust bot detection. We will also consider transaction-based activities as discriminatory features.

## Acknowledgment

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(2014R1A1A1006228). We would like to thank both the reviewers and our shepherd, Guofei Gu, for their insightful comments and feedback which greatly improved the paper.

## References

[1] M. A. Ahmad, B. Keegan, J. Srivastava, D. Williams, and N. Contractor, "Mining for Gold Farmers: Automatic Detection of Deviant Players in MMOGS," in *Computational Science and Engineering International Conference, vol. 4, pp. 340-345*, Aug, 2009.

[2] D. Bethea, R. A. Cochran, and M. K. Reiter, "Server-side verification of client behavior in online games," in *Proceedings of the 17th Network and Distributed System Security Symposium*, 2010.

[3] A. Carbone, G. Castelli, and H. Stanley, "Time-dependent hurst exponent in financial time series," *Physica A: Statistical Mechanics and its Applications*, vol. 344, no. 1, pp. 267–271, 2004.

[4] E. Castronova, "Synthetic worlds: The buisiness and culture of online games," in *University of Chicago Press*, 2005.

[5] ——, "Effects of botting on world of warcraft," http://virtuallyblind.com/files/mdy/blizzard_msj_exhibit_7.pdf, 2007.

[6] E. Castronova, D. Williams, C. Shen, R. Ratan, L. Xiong, Y. Huang, and B. Keegan, "As real as real? macroeconomic behavior in a large-scale virtual world," in *New Media & Society, vol. 11, no. 5, pp. 685*, 2009.

[7] K.-T. Chen, A. Liao, H.-K. K. Pao, and H.-H. Chu, "Game Bot Detection Based on Avatar Trajectory," in *Entertainment Computing ICEC 2008, vol. 5309, pp. 94-105*, 2009.

[8] Y. Chung, C. yong Park, N. ri Kim, H. Cho, T. Yoon, H. Lee, and J.-H. Lee, "Game Bot Detection Approach Based on Behavior Analysis and Consideration of Various Play Styles," in *Journal of ETRI, vol. 35, no. 6, pp. 1058-1067*, Dec. 2013.

[9] A. Conda, "Exclusive: A primer on monetization of f2p games," http://www.alistdaily.com/news/exclusive-a-primer-on-monetization-of-f2p-games, 2012.

[10] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: evidence and possible causes," in *IEEE/ACM Transactions on Networking, vol. 5, no. 6, pp. 835-846*, 1997.

[11] E. Digital, "Group laundered $38m in virtual currencies in 18 months," http://www.engagedigital.com/blog/2008/10/27/group-laundered-38m-in-virtual-currencies-in-18-months/, 2008.

[12] J. Hunter, "The Exponentially Weighted Moving Average," in *Quality Techno, vol. 18, no. 4, pp. 203-210*, 1986.

[13] Infiniti Research Limited, "Global online gaming market 2014," http://www.marketwatch.com/story/global-online-gaming-market-2014-2014-06-25, 2014.

[14] A. R. Kang, H. K. Kim, and J. Woo, "Chatting Pattern Based Game Bot Detection: Do They Talk Like Us," in *KSII Transactions on Internet and Information Systems, vol. 6, no. 11, pp. 2866-2879*, Nov. 2012.

[15] A. R. Kang, J. Woo, J. Park, and H. K. Kim, "Online Game Bot Detection Based on Party-Play Log Analysis," in *Computers and Mathematics with Applications, vol. 65, no. 9, pp. 1384-1395*, May. 2013.

[16] B. Keegan, M. A. Ahmad, J. Srivastava, D. Williams, and N. Contractor, "Dark Gold: Statistical Properties of Clandestine Networks in Massively Multiplayer Online Games," in *Social Computing(SocialCom), IEEE Second International Conference, pp. 201-208*, Aug. 2010.

[17] D. Kushner, "Steamed: Valve software battles video-game cheaters," http://spectrum.ieee.org/consumer-electronics/gaming/steamed-valve-software-battles-videogame-cheaters, 2010.

[18] H. Kwon, T. Kim, S. J. Yu, and H. K. Kim, "Self-similarity Based Lightweight Intrusion Detection Method for Cloud Computing," in *Intelligent Information and Database Systems, Springer Berlin Heidelberg, pp. 353-362*, 2011.

[19] H. Kwon, K. Woo, and H. chul Kim, "Surgical strike: A novel approach to minimize collateral damage to game bot detection," in *ACM NetGames 2013*, Dec. 2013.

[20] J. Lee, J. Lim, W. Cho, and H. K. Kim, "I know what the bots did yesterday: full action sequence analysis using naive bayesian algorithm," in *NetGames'13, Annual Workshop on Network and Systems Support for Games*, Dec. 2013.

[21] S. Mitterhofer, C. Kruegel, E. Kirda, and C. Platzer, "Server-Side Bot Detection in Massively Multiplayer Online Games," *Security Privacy, IEEE*, vol. 7, no. 3, pp. 29–36, May 2009.

[22] A. Roy, M. A. Ahmad, C. Sarkar, B. Keegan, and J. Srivastava, "The Ones That Got Away: False Negative Estimation Based Approaches for Gold Farmer Detection," in *IEEE PASSAT and SocialCom, pp. 328-337*, Sep, 2012.

[23] S. Salihoglu and J. Widom, "GPS: A Graph Processing System," in *Proceedings of the 25th International Conference on Scientific and Statistical Database Management*, 2013.

[24] R. Thawonmas, Y. Kashifuji, and K.-T. Chen, "Detection of MMORPG Bots Based on Behavior Analysis," in *Advances in Computer Entertainment Technology Conference, pp. 91-94*, Dec. 2008.

[25] M. van Kesteren, J. Langevoort, and F. Grootjen, "A Step in the Right Detecting: Bot Detection in MMORPGs using Movement Analysis," in *The 21st Benelux Conference on Artificial Intelligence*, 2009.