Statistical Methods for User and Team Identification in Multiplayer Games

Christopher M. VanderKnyff Darrell J. Bethea Michael K. Reiter Mary C. Whitton*

The University of North Carolina at Chapel Hill

Abstract

We present a new machine learning approach to detect collusion in networked games. Our approach analyzes logs of selected game events as observed at the game server. We tested our approach in multiplayer *Quake III Arena* games, with promising results. In particular, we report the results of applying our approach in both standard game modes and in a game mode that we crafted in order to make collusion detection more difficult. Our approach accurately identified player collusions in all but one game mode, including the one we crafted. We further examine reasons for the failure of our detection mechanism in the cases where it failed.

CR Categories: I.5.2 [Pattern Recognition]: Design Methodology—Classifier Design and Evaluation; H.5.3 [Information Interfaces and Presentation]: Group and Organization Interfaces—Collaborative Computing—Synchronous Interaction; K.8.0 [Personal Computing]: General—Games

Keywords: collusion, gaming, deathmatch, boosting, classification, principal component analysis

1 Introduction

Cheating still remains an endemic problem in online gaming. Players caught cheating are disconnected from their game server and may be banned from reconnecting, but there are typically thousands of servers running for a popular game, reducing the impact of the ban. Additionally, without a global player-ban enforcement system such as those provided by Valve Anti-Cheat [Val 2002], PunkBuster [Eve 2000], or Xbox Live [Mic 2002], the in-game repercussions for being caught cheating are effectively nil. Even the banning systems previously mentioned can be avoided simply by switching accounts. While this may be cost-prohibitive for many, some cheaters have access to compromised accounts and can simply switch identities rather than purchase new copies of their favorite games or subscriptions to Xbox Live. A means of identifying repeat players without reliance on account information would therefore be advantageous.

Collusion is one way by which game state can be conveyed to players who should not ordinarily receive it. To use the popular online game *Counter-Strike: Source* [Val 2004] as an example, players flagged as "dead" in the game become intangible spectators, able to fly throughout the level and watch the surviving players continue to battle. Spectators can give a great tactical advantage to their teammates by conveying the locations of nearby enemies. While in-game chat messages from spectators do not reach active players, the same restriction does not apply to out-of-game communications media such as instant messengers, the telephone, or even speech.

*e-mails: {chrisv, djb, reiter, whitton}@cs.unc.edu

Table 1: *Summary of scoring in Quake deathmatch and CTF modes.*

Game Mode	Victim		
	Suicide	Teammate	Enemy
Deathmatch	-1	n/a	$+1$
Team Deathmatch	-1	-1	$+1$
Capture the Flag	-1	n/a[1]	$+1$

Similarly, colluding players in free-for-all games can increase their scores by sharing information on enemy players or through simple nonaggression pacts.

Our Game Testbed. *Quake III Arena* [id 1999] (hereafter, simply "Quake") was a popular first-person shooter (FPS) in the early 2000s. In it, each player controls an avatar, which runs through a virtual environment firing weapons at the opposition. When an avatar takes a sufficient amount of damage, it plays a death animation and *respawns* (reappears) elsewhere in the game level with full health and no equipment. Upon causing the death of a player, the attacking player scores a point. Players that fall to zero health without damage inflicted by other players (generally through falling or weapon misfires) are considered to have *suicided*, and the suiciding player loses a point. At the end of the round the player with the most kills wins, and the game resets on a different level. This free-for-all game style is known as *deathmatch*. A variation exists wherein players are divided into two teams, Red and Blue, and score points for killing only players of the opposite team; this mode is known as *team deathmatch*. The final game mode provided by retail versions of Quake is called *Capture the Flag* (CTF). CTF games are played on special levels, where players must infiltrate the enemy team's base, steal its flag, and bring it back to their own base to score. Damage from friendly fire is usually disabled in CTF games. The points awarded for each type of kill in these three game modes are shown in Table 1.

New Knowledge. We instrumented the Quake game code to log gameplay events to an XML file. A group of volunteers played deathmatch games using our modified code, and we constructed a multilevel classifier using boosted decision stumps to identify these player profiles in a test game. Our classifier correctly identified players in 65.12 percent of test cases, a statistically significant increase over random selection. Additionally, we tested the efficiency of principal component analysis (PCA) in identifying groups of colluding players in free-for-all and team games. As Quake supports only two teams in its default team game types, we developed a four-team game type with an asymmetric hostility graph ("Paranoia") and tested our algorithm with this mode in addition to team deathmatch and CTF. Our algorithm was 83.33 percent accurate in team deathmatch games, 64.71 percent accurate in CTF games, and 82.35 percent accurate in Paranoia games.

[1] Not possible under usual circumstances. Attacker loses one point when damage from friendly fire is enabled.

2 Previous Work

Behavior Identification. Adaptive boosting, a technique for improving classifier accuracy by chaining weaker classifiers, was initially proposed by Freund and Schapire [1997]. Using boosted decision trees for classification is an active research topic in many other problem domains. Lu *et al.* [2006] explored the use of boosting to identify and enhance Web search queries used for navigating to known Web addresses; their feature space included substring commonalities and frequency of clicks and hypertext anchors. Wang *et al.* [2006] used boosting to identify users biometrically from their palmprints, basing their features on the presence or absence of patterns in the image data. Additionally, Dettling and Bühlmann [2003] examined the efficiency of boosting for tumor classification in genome data, classifying using the position of known gene sequences.

While the boosting meta-algorithm can be implemented with a wide variety of weak classifier techniques, the most popular weak learners are decision trees, specifically the one-level decision trees known as decision stumps [Dettling and Bühlmann 2003]. Several boosting techniques have since been proposed to address weaknesses in the original AdaBoost algorithm. We use LogitBoost, a boosting technique developed by Friedman *et al.* [1998] that uses logistic regression to weight the individual decision stumps. LogitBoost is less susceptible to overtraining than its predecessor and does not overly weight features with low observed variance.

Cheating. Real-time Internet games are subject to a number of popular attack types [Yan and Randell 2005], three of which are skill enhancement, denial of play, and information compromise. Skill-enhancement attacks are best exemplified by aimbots, programs that automatically aim the player's virtual weaponry at on-screen enemies, ensuring significantly better accuracy (and thus a higher score) than that of an average player. Denial-of-play attacks may be literal denial-of-service attacks, such as by flooding the network connections or triggering disconnection of other players, or metaphorical, in which client or server bugs are exploited to render the game unwinnable for others. Information-compromise attacks seek to obtain knowledge of game state not normally available to players, usually by modifying game clients or system infrastructure.

We agree with Webb and Soh [2007] that the general problem of collusion is unsolvable, especially in anonymous Internet games where proctored matches are impractical. A sufficiently subtle group of colluders in an FPS will always be indistinguishable from genuinely skilled (or just lucky) honest players. Spectator proxy systems such as *Half-Life TV* introduce deliberate delays [Otten 2001] into broadcast gameplay to mitigate the effects of cheating. In FPS games, the state of the game typically changes quickly enough for a thirty-second delay to minimize the effects of spectator/player collusion.

Online card games are a favorite domain for anti-collusion research. Vallvè-Guionnet [2005] proposes a collusion detection technique for the card game Botifarra. Yan [2003] discusses techniques for collusion detection and mitigation in contract bridge. In an FPS, however, the only significant interaction between players is the exchange of gunfire. Team members refrain from shooting each other but typically do not hesitate to attack all other players. This shoot/don't-shoot decision manifests itself in the amount of damage done by each player to the others over the course of a match, and we hypothesize that PCA will reveal the underlying team relationships between players.

PCA is a well-studied statistical technique [Harman 1976] we apply to the gaming domain. Given a sample, the basic objective of PCA is to extract the primary factors ("components") that explain the majority of variance observed in the data. The output of PCA is a list of factors ordered by the amount of variance explained, and a correlation matrix linking the input observations to these factors. The first few components generally account for the majority (greater than ninety percent) of observed variance. Successive components contribute diminishing amounts, and are typically ignored. We believe in the case of player damage matrices, the high-orders factors can be interpreted as archetypal team behaviors; the scores of individual team members should align strongly with one of these teams plus a random error factor representing individual variance.

3 Data Collection

We made a number of modifications to the reference Quake code base [id 2005]. Many of these were minor changes to aid our experiment protocols (*e.g.*, per-user configuration profiles for public lab machines running Microsoft Windows), but we also created a new game mode and added logging infrastructure to the game's server code. Data collection was considered a user study by the university and we received Institutional Review Board (IRB) approval before running the games.

3.1 Logging Modifications

We modified the *Quake III Arena* source code to record a subset of gameplay events as they occur on the server. While Quake already provides similar functionality in the form of demo recording, demos in Quake are recorded from the perspective of the client. This is disadvantageous for our purposes for two reasons. First, the client typically operates with a much shorter time step than the server. To ensure smooth in-game animation, the client interpolates player and entity positions using a dead-reckoning algorithm. Rendered player positions may therefore be inaccurate when compared to the authoritative server state, additionally causing "warping" effects if the client must resynchronize frequently. A client-side configuration option called "g_synchronousClients" can be used to disable dead reckoning [id 2005], but this causes a noticeable degradation in the simulation frame rate, which has been linked to decreased performance in other applications [MacKenzie and Ware 1993]. Second, Quake demos are generated from network messages received by the client. For security and performance reasons, a player's state is included in updates only for nearby players. Distant events are not transmitted to other clients, so recorded demos are not representative of the entire game except in very small levels.

To overcome these limitations, we record client events from the authoritative copy of the game state held by the server. Rather than store events as opaque network messages, we use an XML-based format to aid interpretation. We store player positions and orientations for each frame of the simulation, which typically updates every fifty milliseconds. In addition, we record server configuration variables at server startup, time of game termination, and timestamped discrete events such as player injuries as they occur. As the overwhelming majority of data associated with each event is numeric, conversion to canonical XML format is trivial; the performance impact of writing these data as XML tags is negligible. (See the appendix for details on our logging format.)

3.2 New Game Mode: Paranoia

As a first step toward general collusion detection, we analyzed team games of Quake. Here the colluding factions are obvious, unchanging, and easily verifiable (we disallowed switching teams during data collection). One drawback to testing collusion-detection algorithms on popular FPS games is that most team game designs

Table 2: *Points scored by a member of each team for killing members of each other team in Paranoia mode.*

		Victim			
		Red	**Blue**	**Green**	**Yellow**
Attacker	**Red**	−1	+1	−1	0
	Blue	0	−1	+1	−1
	Green	−1	0	−1	+1
	Yellow	+1	−1	0	−1

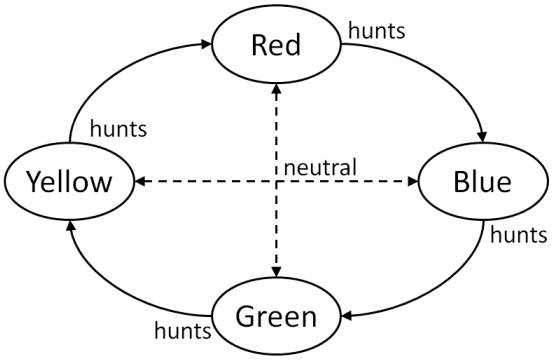

Figure 1: *Diagram showing relationships between the four teams in Paranoia.*

(Quake included) involve only two factions: Red and Blue, Axis and Allies, humans and aliens. When additional factions exist, the basic gameplay remains essentially unchanged; enemy uniforms simply come in more than one color. Naïve approaches to collusion detection may therefore not scale well to games with more than two distinct factions.

We added a four-team deathmatch variant largely inspired by *The Ship* [Out 2006] which we dubbed "Paranoia." This mode introduces three twists to the standard team deathmatch gameplay. First, each team scores points only by killing members of one specific other team, and loses points for killing friendlies (team members) and neutrals (see Table 2). Second, the hunter/prey relationship is not symmetric, as shown in Figure 1. The Blue team is assigned to hunt members of the Green team, while Blue is itself hunted by Red. Players who kill their hunters score no points: they are neither directly rewarded nor otherwise penalized. They are, however, still alive to hunt their own prey. Third, players receive imperfect information on other players' team affiliations. From a player's perspective, only two teams are instantly and reliably identifiable on sight: their own team, and their prey. Members of both the team hunting them and the neutral team receive a default, ambiguous appearance (see Figure 2). Players must therefore rely on behavioral cues to assess hostility and identify enemies.

This new gameplay style is designed to hamper naïve attempts at team identification in several ways. The increased number of teams prevents players from simply being categorized as "Blue" and "Not Blue," with the latter being identically equal to the only other team. In addition, team games are typically not characterized by large amounts of fire directed at teammates or noncombatants. By eliminating the visual distinction between neutral and hostile players, we enabled a subtler style of play characterized by deception and

Figure 2: *Members of the (from left) Red, Blue, Green, and Yellow teams from the perspective of a Red player.*

increased the amount of uncertain weapons fire[2] exchanged. We observed that some players, when low on health, would occasionally seek out a presumed neutral player (*i.e.*, not a current attacker) and behave aggressively through continued following (as if tailing the other player) and shooting to miss. The second player would frequently interpret this behavior as an attack from the other team, and retaliate, killing the first (neutral) player and losing a point. Conversely, we observed some hunters low on health or ammunition disguising themselves as neutrals by ignoring their prey.

3.3 Participant Demographics

Fifty men and four women played games of Quake in one of our building's computer labs using our modified client and server. Thirty-nine participants were graduate students at the time of play, eleven were undergraduates, two were professors, and the remaining two were members of the public. All players were recruited via postings to mailing lists or through word of mouth. The median age for participants was twenty-four. Each participant also self-rated his experience with first-person games (including games other than Quake, such as *Halo* or *Team Fortress 2*) on a scale from one ("I never play FPS games") to five ("I play FPS games daily"). The results of this survey are shown in Figure 3. After each session, participants gave feedback on the games they played.

4 Results

4.1 User Identification

Five players and two of Quake's built-in artificial intelligences ("bots") played four rounds of deathmatch on the same game level. The first three rounds of gameplay were used to train the classifier; the fourth round was withheld for validation. To construct a classifier for player identity, we delimited each prerecorded gameplay sequence along player lifespans, *i.e.*, the time from (re)spawning to death. Rather than analyze motion profiles, we computed a number of gameplay features from logged events associated with each player. These features were selected by applying expert knowledge of typical player behaviors. The precise feature space measured for the classifier may be seen in Table 3. Upon player death or server termination, the feature vector was appended to the sample data set

[2]We refer to weapons fire where the attacker is uncertain whether his target is a member of a hostile team.

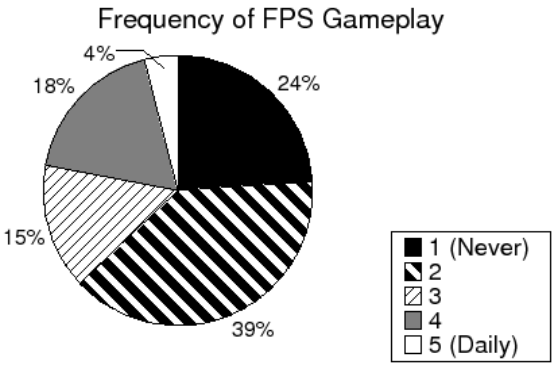

Figure 3: *Self-reported frequency of play of Quake-like games among participants.*

and then reset for the next life. Each life vector was then labeled—simply the player's nickname if human, or "AI" if a bot—and input into the classifier.

In total, the training data set used to construct the classifier (three 20 minute games) included 794 feature vectors distributed among the six player classes. The verification data (a single 20 minute game) yielded eighty-six feature vectors for analysis. We then constructed a multilevel LogitBoost classifier from the training data following the procedure described by Dettling and Bühlmann [2003]. We then tested the (default) twenty-round classifier against the validation data. The classifier correctly identified fifty-six player profiles and missed thirty, yielding an overall accuracy of 65.12 percent. Pearson's chi-square test shows a strongly statistically significant difference between our algorithm and naïve random association of player to feature vector. In this case, $\chi^2 = 150.5$, with 1 *df*, $p < 0.001$.

4.2 Team Identification

We used two games of Capture the Flag, four games of team deathmatch, and seven games of Paranoia for this work. For each n-player game, we created an $n \times n$ matrix where cell (i, j) is defined as the number of damage points player i inflicted upon player j during the game according to ⟨player-hurt⟩ events in the game log. Cells along the diagonal (*i.e.*, cells (i, i)) represented self-damage from weapon misfires, falling, and hazardous level features. Players did not change teams or drop out during play, so each row of the matrix is the result of allegiance to a single team. We then performed PCA on each matrix, selecting the minimum number of factors necessary to explain ninety percent of observed variance. After obtaining an initial factor set from PCA, we used varimax rotation to maximize factor loadings as discussed in Harman [1976]. All statistical procedures were done using SAS 9.1.3. We then verified our results by comparing the reported factor loadings for each player to the team he joined during the game.

Our technique's accuracy in the three game types is shown below in Table 4. In a k-team game ($k = 2$ for team deathmatch and CTF, $k = 4$ for Paranoia), a player's team affiliation is considered correctly identified for our results if the following conditions are met: First, the player's damage vector must be strongly ($|r| > 0.65$) correlated to one and only one of the first k principal components. Second, the majority of the other players strongly correlated to the player's factor must be the player's actual teammates during the game. In the PCA results from the damage matrix, it typically took more than two factors to account for ninety percent of observed variance. In order to evaluate the discriminative power of our second criterion, we performed the PCA technique a second time in these cases, but took only the first two principal components regardless of the amount of variance explained (average eighty-six percent of variance for team deathmatch, sixty-nine percent for CTF). The results of these secondary tests are shown as the "forced" game types in the table.

Finally, we recorded six deathmatch games with six players each. Three of these games each featured a pair of players in collusion, and the other three were true free-for-alls used for comparison. These colluding players were not hostile toward one another, though to allay suspicion they did not otherwise act as a team. PCA was unsuccessful at detecting this collusion, generally assigning the observed players to factors with no obvious pattern. There were no highly-correlated factors in common between the colluding players.

5 Discussion and Future Work

User Identification. We consider our boosting technique to be effective in discriminating between player profiles stored on a deathmatch server. Construction of the classifier from sample data took approximately two minutes on an Athlon 64 with two gigabytes of RAM. Further code optimization can probably speed classifier construction, which will be beneficial as multilevel LogitBoost scales linearly with both class count and training set dimensionality. Once constructed, however, the decision stumps that make up the classifier can be evaluated trivially; our implementation required no more than fifty milliseconds per classification.

Further inspection of the user identification results revealed that a majority (63.33 percent) of the erroneous classifications occurred in cases where insufficient data were provided to the classifier due to abnormally short lives. Using player lives to delimit feature vectors magnifies the data's existing variance in player ability: good players live longer and inflict damage more often, so many of the per-life feature totals are higher for good players than bad. Even the best players, however, are subject to misfortune. Battling players may be near the randomly-chosen respawn point and can kill the newly respawned player in the crossfire. The victim's log for this lifespan will be short, including very few item pickups and zero accuracy with his favorite weapons. Such values are more typical of low-skill players, causing this particular feature vector to be misclassified as that of a novice player.

Future in-game player classification algorithms should use a sliding-window approach instead, examining player performance over the past n seconds rather than over the past life. In addition to providing more continuous feedback (as expert players can often go for several minutes without dying, especially in CTF scenarios), this new feature extraction process should be less sensitive to such variations in observed player skill. Another direction for future work is improvement of the feature selection process. While easy to calculate from a prerecorded data file, our feature space does not take movement patterns into account beyond the simple count of room changes. Some people who played in the Paranoia mode reported an ability to identify frequent opponents based solely on their movement patterns and modified their strategy accordingly. Consequently, further refinement to the feature space used would be warranted, especially inclusion of maneuvering patterns.

Team Identification. Analysis of team deathmatch games generally revealed three principal components rather than the expected two. The first two factors corresponded well to the two teams present, with each factor exhibiting high positive correlations for most of the respective teammates and weak correlations for everyone else. In each case, however, a third factor correlated strongly

Table 3: *List of features used in player classification.*

Feature	Events Used	Notes
Lifespan in seconds	player-spawn, player-kill	—
Points scored	player-score	—
Source of death	player-kill, server-terminate	Whether killed by enemy, suicide, or server shutdown
Weapon preferences	weapon-fire	Proportions of damage dealt with each weapon
Weapon accuracies	weapon-fire, player-hurt	Percentages of hits for each weapon
Total damage inflicted	player-hurt	Includes self-damage from explosives
Proportion of splash damage inflicted	weapon-fire, player-hurt	Refers to explosive damage not inflicted by direct hits
Total damage received	player-hurt	Includes self-damage and falling
Maximum damage received	player-hurt	Over a single simulation frame
Average shot damage multiplier	weapon-fire	Affected by possession of Quad Damage
Number of room transitions	player-room-change	—
Number of jumps	player-jump	—
Total time spent crouching	duck-begin, duck-end	—
Number of healing items picked up	item-pickup	Includes both health and armor
Average value of healing items	item-pickup	For each of health and armor
Number of weapon items picked up	item-pickup	Includes both weapons and ammunition
Number of powerup items picked up	item-pickup	Examples: Quad Damage, personal teleporter

Table 4: *Accuracy of principal component analysis in identifying team affiliations. Significant tests are shown in boldface.*

Game Type	Number of Games	Factors Identified	Correct Guesses	Incorrect Guesses	Overall Accuracy	Chi-Squared Test (Vs. Random Selection)
Team Deathmatch	4	3	30	6	83.33%	$\boldsymbol{\chi^2 = 16.8, df = 7, p < 0.019}$
Capture the Flag	2	4	11	6	64.71%	$\chi^2 = 4.5, df = 3, p < 0.213$
Paranoia	7	4	56	12	82.35%	$\boldsymbol{\chi^2 = 36.4, df = 13, p < 0.001}$
Team DM *forced*	4	2	34	2	94.44%	$\boldsymbol{\chi^2 = 28.8, df = 7, p < 0.001}$
CTF *forced*	2	2	14	3	82.35%	$\boldsymbol{\chi^2 = 9.2, df = 3, p < 0.027}$

with those players not linked to the other factors. The loadings for this factor typically showed the previously unclassified players: players on one team had high positive correlations, and the players on the other team had high negative correlations. For these extra players, inspection revealed no strong correlation to either of the two "team factors." While the sign change in correlation correctly indicated opposition between the two players during the game, we do not consider our analysis technique to have correctly identified team affiliations for the players correlated with factors other than the first two. Visual inspection of the logs indicated these players typically fought each other more than the other enemies present, probably contributing to their damage totals being organized around an extra axis.

Similarly, CTF games typically revealed five principal components rather than two. These additional factors exhibited similar characteristics to the extraneous factors in team deathmatch as described above. We hypothesize, and visual inspection of the gameplay logs supports, that in fact a four-team analysis of CTF may be valid. In this game type, players typically divide themselves into "offense" and "defense" roles. Offensive players raid the enemy base in order to capture their flag, and defensive players stay near the flag to prevent this from happening. As a result, defensive players typically engage only the enemy attackers, and rarely come into contact with enemy defenders. Defender/defender fights typically occur only when a flag is being captured, as the entire team tries to kill the enemy flag carrier and stop the capture. Visual inspection of recorded player behaviors (e.g., proximity to friendly base) confirms the division between offense and defense roles for these players.

Paranoia game logs were generally categorized accurately, but would frequently add an extra factor with one affiliated player. In five of these six cases, the player so categorized was a total novice ("Never" played FPS games) and attained a much lower score than his teammates. Given the relative lack of team coordination these novices usually exhibited, this negative result is perhaps unsurprising.

We find that PCA is a useful technique for detecting groups of players directly collaborating or competing in team games. The input data are easy to obtain from the game logic, and anecdotal evidence suggests similar results may be obtainable from player kill counts rather than inflicted damage totals, though this approach would be less tolerant of uneven player skill.

We do not, however, consider our PCA-based technique suitable for detecting isolated instances of collusion. Colluding players had no highly-correlated principal components in common, and player-factor correlations were typically weak ($|r| < 0.65$). This result may stem from the basic structure of deathmatch games; with n players, each with a differing agenda, there are too many player behavior archetypes (i.e., everyone against player one, everyone against player two, and so on) for the available set of observations. Principal component analysis seeks factors that explain multiple observations at once, which is unlikely to map well to player agendas in such an individualist scenario. Additionally, the level we used for our deathmatch experiment contained hazardous environmental features that resulted in a higher-than-usual proportion of player suicides. Ignoring self-damage before performing PCA appears to greatly improve the accuracy of the collusion detection results, and we plan to verify this observation in future work.

Acknowledgements

The authors wish to thank our funding agencies: the Office of Naval Research (VIRTE Project), the NIH National Institute for Biomedical Imaging and BioEngineering, and SAIC. Additionally, we thank Leonard McMillan for his initial suggestion to use boosting.

References

DETTLING, M., AND BÜHLMANN, P. 2003. Boosting for tumor classification with gene expression data. *Bioinformatics 19*, 3, 1061–1069.

EVEN BALANCE, INC. 2000. *Punkbuster*. http://www.evenbalance.com/.

FREUND, Y., AND SCHAPIRE, R. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*, 1, 119–139.

FRIEDMAN, J., HASTIE, T., AND TIBSHIRANI, R. 1998. Additive logistic regression: A statistical view of boosting. *Annals of Statistics 28*, 2, 337–407.

HARMAN, H. 1976. *Modern Factor Analysis*. University of Chicago Press.

ID SOFTWARE. 1999. *Quake III Arena*. http://www.idsoftware.com/games/quake/quake3-arena/.

ID SOFTWARE. 2005. *Quake III Arena 1.32b Source Code*. ftp://ftp.idsoftware.com/idstuff/source/quake3-1.32b-source.zip.

LU, Y., PENG, F., LI, X., AND AHMED, N. 2006. Coupling feature selection and machine learning methods for navigational query identification. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management*, ACM, 682–689.

MACKENZIE, I. S., AND WARE, C. 1993. Lag as a determinant of human performance in interactive systems. In *INTERCHI '93: Proceedings of the INTERCHI '93 conference on Human factors in computing systems*, ACM, 488–493.

MICROSOFT CORPORATION. 2002. *Xbox Live*. http://www.xbox.com/live/.

OTTEN, M. 2001. Broadcasting virtual games in the internet. Tech. rep., Valve Corporation, http://www.slipgate.de/download/BroadcastingVirtualGames.pdf.

OUTERLIGHT LTD. 2006. *The Ship*. http://www.theshiponline.com/.

SCHAPIRE, R. E., AND SINGER, Y. 1998. Improved boosting algorithms using confidence-rated predictions. In *COLT '98: Proceedings of the 11th Annual Conference on Computational Learning Theory*, ACM, 80–91.

SCHLUESSLER, T., GOGLIN, S., AND JOHNSON, E. 2007. Is a bot at the controls?: Detecting input data attacks. In *NetGames '07: Proceedings of the 6th ACM SIGCOMM Workshop on Network and System Support for Games*, ACM, 1–6.

VALVE CORPORATION. 2002. *Valve Anti-Cheat*. http://www.steampowered.com/.

VALVE CORPORATION. 2004. *Counter-Strike: Source*. http://www.counter-strike.net/.

VALLVÈ-GUIONNET, C. 2005. Finding colluders in card games. In *ITCC '05: Proceedings of the International Conference on Information Technology: Coding and Computing - Volume II*, IEEE Computer Society, 774–775.

WANG, X., GONG, H., ZHANG, H., LI, B., AND ZHUANG, Z. 2006. Palmprint identification using boosting local binary pattern. In *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*, IEEE Computer Society, 503–506.

WEBB, S. D., AND SOH, S. 2007. Cheating in networked computer games: a review. In *DIMEA '07: Proceedings of the 2nd International Conference on Digital Interactive Media in Entertainment and Arts*, ACM, 105–112.

YAN, J., AND RANDELL, B. 2005. A systematic classification of cheating in online games. In *NetGames '05: Proceedings of the 4th ACM SIGCOMM Workshop on Network and System Support for Games*, ACM, 1–9.

YAN, J. 2003. Security design in online games. In *ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference*, IEEE Computer Society, 286.

Appendix: Logged Events

Our logging infrastructure records the Quake gameplay events listed below. Positions are recorded as X/Y/Z triplets in the game engine's internal coordinate system. Orientations are recorded in Euler angles (yaw, pitch, roll) using degrees. All recorded events are timestamped with the number of milliseconds since sever initialization.

Server Events

server-init Server initializes.
Details: Server configuration string used by Quake

server-term Server terminates, through match conclusion or operator shutdown

player-join Player connects to the server, switches teams, or enters spectator mode
Details: Player nickname, new team name

player-quit Player disconnects from the server
Details: Player nickname

Player Behavior Events

player-spawn Player respawns in the world
Details: Player nickname and new position

player-move Player is alive and server simulation state has updated (every 1/20th of a second)
Details: Player nickname, position, and orientation

item-pickup Player picks up an item (*e.g.*, weapons, health, ammunition, armor, or special powerups like the Quad Damage or personal teleporter)
Details: Player nickname, item type

player-jump Player jumps
Details: Player nickname

duck-begin Player begins crouching
Details: Player nickname

duck-end Player stops crouching
Details: Player nickname

Weapon Events

weapon-change Player switches weapons
Details: Player nickname and new weapon type

weapon-fire Player fires a weapon[3]
Details: Player nickname, weapon type, and damage multiplier (usually 1.0, but can be increased by the Quad Damage and Doubler powerups, or decreased by voluntary player handicap[4])

player-kill Player killed by another player or suicide
Details: Killer nickname (or "*nobody*" for falling/environmental damage), victim nickname, and means of death (*e.g.*, shotgun, falling, rocket launcher)

player-hurt Player takes damage from a single source
Details: Inflictor nickname, victim nickname, health damage applied, armor damage applied, and damage type (see player-kill, "means of death") if damage is fatal

Scoring Events

player-score Player's personal score changes
Details: Player nickname and new score

team-score An overall team score (Red, Blue, Green, or Yellow) changes
Details: Team and new score

flag-steal A flag has been stolen (CTF only)
Details: Nickname of player now carrying the enemy flag

flag-dropped A carried flag is dropped (CTF only)
Details: Team owning (not the team formerly carrying) the dropped flag

flag-returned A dropped flag returns to its owners' base (CTF only)
Details: Team owning the returned flag

Communication Events

chat-broadcast Player sends a text message to all other players
Details: Player nickname, chat message

chat-teamcast Player sends a text message to other players on his team
Details: Player nickname, chat message

chat-unicast Player sends a text message to one specific other player[5]
Details: Sending player nickname, receiving player nickname, chat message

[3]Because of the internal implementation details of Quake's gauntlet weapon, we record weapon-fire events for the gauntlet only in frames where the gauntlet-wielding player inflicts damage.

[4]No players used handicaps in the experiment.

[5]No player did this during the experiment, though Quake's AI will occasionally send messages this way.