A Peer Auditing Scheme for Cheat Elimination in MMOGs

Josh Goodman              Clark Verbrugge
jgoodm7@cs.mcgill.ca    clump@cs.mcgill.ca
McGill University
School of Computer Science
Montréal, Canada
Table of Contents

- Introduction
- Hybrid Solution
- Design
- Results
- Conclusion
Introduction: Cheating Impact

- Cheating in MMOGs can have an **important** impact
- Example: cheaters **banned** for using the “Movement Enhancing Hack” in Final Fantasy XI
- There is a FFXI cheating task force

- Introduction  •  Hybrid Solution  •  Design  •  Results  •  Conclusion
Introduction: *Current Solutions*

- Current Cheat Elimination Solutions are:
  - Manual:
    - Log reviewing
    - Complaint based
  - Methods that focus on a specific cheat
  - Using to Client Server (C/S) models
- **But:** harder to implement and limit scalability (C/S over P2P)
Introduction: Other Solutions

- Automatic, scalable cheat resistance is very desirable, however:
  - Cheating domain: it is hard to define exactly what “cheating” is
  - Performance: a solution must be scalable, having low overhead
  - Accuracy: a solution should punish only cheaters
    - Should avoid mistaking a trustworthy client as a cheater

False positives
Introduction: *Motivating Example*

- McGill MMOG Testbed: *Mammoth*
- Problem:
  - Path-finding done *client side*
  - Allows for abuse / cheating
- Example:
  - Normally, Bob finds the path leading to the destination
Introduction: Motivating Example

- McGill MMOG Testbed: *Mammoth*
- Problem:
  - Path-finding done **client side**
  - Allows for **abuse / cheating**
- Example:
  - **Bob** can also cheat sending a path that ignores obstacles
Introduction: Motivating Example

- Alternate approach
  - Path-finding done server side
  - Lowers chances for abuse / cheating
  - Path-finding is expensive
  - Can cause a bottleneck

- Idea: Marry both approaches
  - Use **P2P** for load management
    - Use **Peers** to resolve path requests
  - Use **C/S** for cheat resistance
    - Use **server** as an arbiter
Hybrid Solution: The IRS Model

• Approach MMOGs with a Hybrid Model
  – Try and create a network model that is the best of both worlds

• The IRS hybrid model:
  – Uses a centralized server for verification / persistence
  – Uses P2P communication for message handling

• Goal:
  – Reduce the occurrence / accessibility of Cheating
  – Reduce the computational requirements of the Server
Hybrid Solution: *Cheat Detection*

- Detection of suspicious behaviour
  - Use *peer auditing*
    - Send copies of requests to an extra client
    - Compare both answers
  - If both answers are the same
    - Assume they are both correct
  - If both answers differ
    - Assume either is cheating
    - Compute the **true** result and compare both answers
Hybrid Solution: *Cheater Identification*

- There are many causes for suspicious behaviour
  - Hardware differences
  - Communication failure
  - Cheating

- Differentiating between errors and cheating:
  - Use a *Trust Metric*:
    - Group the failures by severity
    - Count the number of failures against successes
    - Since random hardware or communication errors are rare
    - Use this to determine if a client is likely cheating
Hybrid Solution: Summary

- We propose the IRS model as a cheat reduction solution that is:
  - *Scalable with Low overhead*: allows P2P communication and reduces server **CPU load**
  - *Automatic*: peer auditing allows for the identification of suspicious behaviour
  - *Accurate*: Trust based scoring differentiates between random *errors* and *cheating behaviour*
Design: Overview

- The IRS Model incorporates the following:
  - Communication Model
  - Message Verification Scheme
    - Auditing
    - Monitoring
    - Quick Testing
  - Trust method for identifying cheaters
  - Disciplinary system that removes malicious clients
Design: Components

• Components of the IRS Model:

  - **Server**
    - Acts as arbiter for clients
    - Manages gamestate
    - Handles Login

  - **Monitors**
    - Owned by the game providers
    - Monitor and verify audits

  - **Clients**
    - The game players
    - Acts as a *proxies*
    - Has a client *proxy*
Design: Load Distribution Protocol

- The IRS model's load distribution protocol is:
  - P2P oriented:
    - Proxies are clients that compute message results for others
    - Each client has a proxy and acts as a proxy for others
  - C/S oriented:
    - Server handles login
    - Result monitoring
    - Gamestate maintenance
    - Message relaying
    - Matching clients and proxies
Design: *Load Distribution Protocol*

- 4 Protocol Phases (Server)
  - 1. Proxy Assignment
    - Randomly matches clients to proxies
    - Proxies are assigned by server at regular intervals

![Diagram of proxy assignment process](image)
Design: Load Distribution Protocol

- 4 Protocol Phases (Server)
  - 2. Message Relaying
    - Server relays path finding requests from a client to its proxy
    - The proxy is responsible for resolving said request
Design: *Load Distribution Protocol*

- **4 Protocol Phases (Server)**
  - **3. Peer Auditing**
    - The server randomly audits clients by simultaneously sending the request message to an extra client (co-auditor)
Design: Load Distribution Protocol

- 4 Protocol Phases (Server)
  - 3. Peer Auditing
    - The proxy's message is quick-tested and forwarded
    - The server then compares both resolved messages

Diagram:

- Server
- C1
- C4
- C5

Compare both resolved messages

- Introduction • Hybrid Solution • Design • Results • Conclusion
Design: Load Distribution Protocol

- 4 Protocol Phases (Server)
  - 3. Peer Auditing
    - If the comparison fails, the audit is sent to the monitor
Design: Load Distribution Protocol

- 4 Protocol Phases (Server)
  - 4. Message Handling
    - Quick Testing of resolved messages
    - Relaying the resolved message to appropriate clients

If successful the message is returned to C₁ and other interested clients.
Design: Load Distribution Protocol

- 4 Protocol Phases (Server)
  - 4. Message Handling
    - Quick Testing of resolved messages
    - Relaying the resolved message to appropriate clients
Design: Auditing Scheme

• Peer Audits
  – Examine resolved messages returned by proxies
  – Started randomly
  – Opened during the message relaying phase
  – Compared at a later time

• Audits yield the following:
  – Identical
  – Equivalent
  – Inequivalent
  – Infeasible
Design: *Comparison Types*

- **Identical:**
  - All points are coincidental
  - This is the *best* possible comparison result.
Design: Comparison Types

• Equivalent:
  - Same *starts* point
  - Same *ends* point
  - Similar *lengths*
  - Regarded as a *positive* result
Design: *Comparison Types*

- **Inequivalent:**
  - Different **start** points or
  - Different **end** points or
  - Dissimilar **lengths**

- Regarded as a **negative** result
Design: Comparison Types

- **Infeasible:**
  - Violates *game rules*
  - Passes through obstacles or
  - Leads to inaccessible areas
  - This is the **worst** possible comparison result

*Image of a game map showing a start and end point.*
Design: Monitoring

- Failed audits are subjected to Monitoring
  - Monitors are controlled by game company
  - Monitors resolve the original request message
  - Compares its result to the two results contained in the audit
  - Determines which clients are responsible for the audit failure
Design: Trust

- **Trust**
  - Designed to distinguish cheats and error
  - History based
  - Identical and equivalent messages cause an increase
  - Inequivalent and infeasible messages causes a drop
  - Can require a discount factor in order to forget older infractions
Design: Quick Testing

- *Quick Testing* eliminates worst-case inaccuracies
  - Computed cheaply
  - Can only determine if a message is infeasible or not
  - Is used before relaying messages back to clients
  - If failed, the server will compute its own resolved message
Design: Disciplinary Action

- Disciplinary Action
  - **Boot**ing: when an inaccuracy is caught
    - Temporary
    - Early warning
    - Breaks up consecutive cheating
  - **Ban**ning: when trust falls below the *ban threshold*
    - Permanent
    - Ultimate deterrent
    - Lowers the number of cheaters in the system
Results

• Cheat reduction tests in 2 environments
  – Static client base
  – Dynamic client base

• Load analysis
  – Determine CPU load reduction
  – Bandwidth increases
  – Costs of cheat reduction
Results: *Parametrization*

- **Client Simulation:**
  - **Legit Clients:**
    - Trustworthy clients
    - Never attempt to cheat
    - Have a small chance to fail
  - Are 99% accurate
Results: *Parametrization*

- Client Simulation:
  - Griefers:
    - Want to disturb others
    - Cheat in order to ruin other's game play
    - Example: sending clients in the wrong direction
  - Will “grief” 50% of the time, returning inequivalent results
Results: *Parametrization*

- Client Simulation:
  - Hackers:
    - Malicious clients
    - Attempt to destabilize the game
    - Example: returning a result with a different start point in order to "teleport"
  - Will cheat 50% of the time
    - 50% of said cheats will be infeasible
    - The other 50% will be inequivalent
Results: *Parametrization*

- **Client Simulation:**
  - Monitors:
    - Owned by the game providers
    - Used to monitor audits after the fact
  - Assumed to resolve messages 100% accurately
  - Compares its result to audit
  - Determines which client is responsible
Results: *Parametrization*

- **Client Overview:**
  - Legit Clients: 99% accurate, 1% error
  - Hackers: 50% accurate, 25% inequivalent, 25% infeasible
  - Griefers: 50% accurate, 50% inequivalent
  - Monitors: 100% accurate
  - Clients make requests every ~[0,3] seconds
    - Based on practical game data
Results: *Parametrization*

- **Cheat Reduction:**
  - Audit: 10% of requests, Monitor: 5% of positive audits.
  - Boot time: 30 secs
  - Ban threshold: -15
    - Determined as best candidate experimentally
  - Trust metric:
    - Also Determined as best candidate

\[ T = [\text{identical}] + [\text{equivalent}] - [\text{inequivalent}]^{1.5} - [\text{infeasible}]^2 \]

An exponent of 1.5 causes less serious cheats to ramp up quickly, but not too quickly as to effect legit clients.
An exponent of 2 causes more serious cheats to ramp up exceedingly quickly removing malicious clients effectively.
Results: *Experiment 1*

- Experiment in a **static** setting
- Initial clients:
  - 8,500 legit
  - 750 hackers
  - 750 griefers
- 20 minute experiments
- Very few false positives ~ 0.4 clients per experiment
Results: Experiment 1

- Experiment in a **static** setting
- Initial clients:
  - 8,500 legit
  - 750 hackers
  - 750 griefers
- 20 minute experiments
- Very few false positives ~ 0.4 clients per experiment

![Average Ratio of Cheat Messages-In vs Time (Seconds)](chart.png)
Results: *Experiment 2*

- Experiment in a **Dynamic** setting
- Initial clients: 0
- Per second:
  - 6 legit
  - 2 hackers
  - 2 griefers
- 60 minute experiments
- More false positives ~ 8 per experiment

![Average Number of Cheating Clients](chart)

- Introduction  •  Hybrid Solution  •  Design  •  Results  •  Conclusion
Results: *Experiment 2*

- Experiment in a **Dynamic** setting
- Initial clients: 0
- Per second:
  - 6 legit
  - 2 hackers
  - 2 griefers
- 60 minute experiments
- More false positives ~ 8 per experiment
Results: *Rate of Cheating Analysis*

- **Formal analysis**
  - Relates rate of cheating to expected ban time
  - Shows:
    - A cheater must reduce its rate of cheating to last
    - A lower rate of error extends game time drastically
    - A client with a 0.1% error rate is expected to last will last ~7.5 months of continual gameplay
Results: *Experiment 3a*

- Experiment on Load/Overhead
- From static experiment data
- 60 minute experiments
- C/S results depict a load of around 250,000-275,000 units.

![Server Load Graph](image-url)
Results: *Experiment 3b*

- Experiment on Load/Overhead
- From static experiment data
- 60 minute experiments
- Compares
  - C/S
  - IRS w/ audits
  - IRS w/o audits

```
Results: Requests Per Client

Audits
No-Audits
C/S

Time (Seconds)
```

- Introduction
- Hybrid Solution
- Design
- Results
- Conclusion
Conclusion: Summary

- Trade-off between **scalability** and **Cheat Resistance**
- IRS model shows
  - Good CPU load reduction ~ 10%
  - Ability to eliminate cheaters quickly
    - In approximately 400 seconds (due to booting)
  - Higher bandwidth > 200%
  - Higher Number of Hops > 200%
Conclusion: *Future Work*

- The examination of models which:
  - Ensure the IRS cheat reduction guarantees
  - Lower bandwidth cost
  - Lower latency

- The examination of auditing systems which:
  - Use adaptive auditing based on trust

- Integration of the IRS model into Mammoth
  - Alleviate cost of server side path-finding
  - Investigate IRS properties in a concrete setting
References


[3] SQUARE ENIX, Final Fantasy XI.


References


References


References


References


References


References


Example Bullet Point slide

• These templates are for personal use only and must not be distributed, sold or displayed on the web by anyone other than Presentation Helper.

• Bullet point
  – Sub Bullet
Design: Communication Model

• 4 Communication Phases (Server)
  – 1. Proxy Assignment
    • Done by server at certain intervals
  – 2. Message Relaying
    • Server relays messages from a client to its proxy
    • The proxy is responsible for resolving said message
  – 3. Peer Auditing
    • Resolved messages computed by different clients on identical requests are compared
  – 4. Message Handling
    • Quick Testing of resolved messages
    • Relaying the resolved message to appropriate clients
Design: Communication Model

- Diagram of phases 2-4:
Introduction: Current Solutions
Introduction: Cheating Focus

- In MMOG's there are a vast variety of cheating behaviours
- It is also difficult to formulate a precise definition of cheating
- Many ad hoc cheat elimination systems exist
- However, with P2P communication avoiding abuse of authority is imperative
- Therefore: we focus on reducing/eliminating abuse of authority cheats