Cheat-Proof Playout for Centralized and Distributed Online Games

By

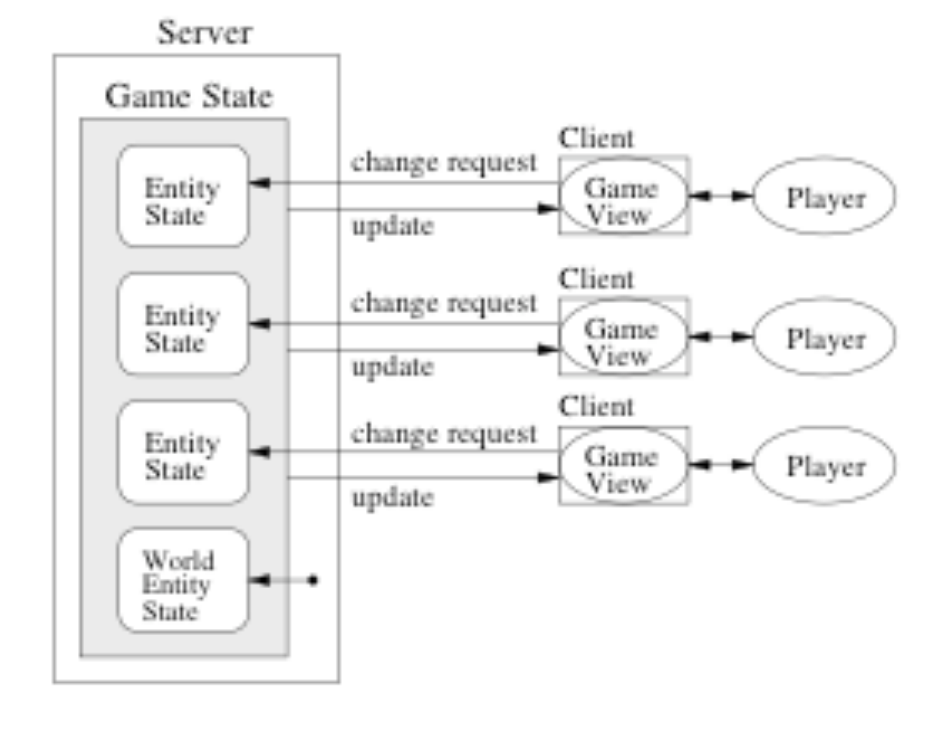
Nathaniel Baughman and Brian Levine

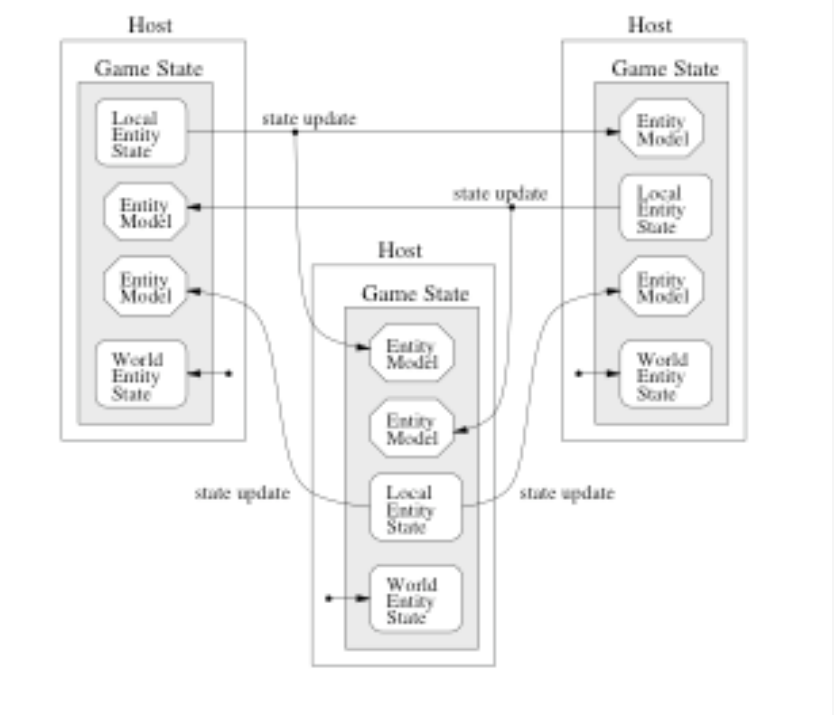
(danny perry)

Game System - Assumptions

- Grant Cheaters:
  - Read, insert, modify, and block network messages
  - Game software is readable by user - including **game state** and **cryptographic keys**
  - Which means security by obscurity won't work.

Game System - Centralized or Distributed





Common Synchronization Techniques Detrimental

- Example 1: Dead Reckoning
  - Potential for cheating - “suppress-correct cheat”
- Example 2: Stop-and-wait Simulations
  - Potential for cheating - “lookahead cheat”

Dead Reckoning-suppress-correct cheat

- Assume n steps in game simulation allowed to be dead-reckoned before connection is lost
- What if we drop n-1 packets?
- Then we have an up to date view of the world, but everyone else has an old view of us

- Anyone remember their proposed cheat?

Dead Reckoning-suppress-correct cheat

- Cheat Scenario:
- Player S chases player A
- Player S drops n-1 updates
- Meanwhile A uses dead reckoning to guess, but never REALLY knows where S is.
- S sends fabricated nth update that places S right behind A

Dead Reckoning-suppress-correct cheat

- As long as S sends a plausible update every nth step, A cannot differentiate cheating from sluggish link

- “We can conclude that fair play is indistinguishable from cheating when dead reckoning is used.” (p. 4)

Stop-and-wait Simulations - lookahead cheat

- Stop-and-wait simulations like AoE
- Eliminates possibility of “suppress-correct cheat” because there is no dead reckoning
- However stop-and-wait simulations allow for another kind of cheat

- Anyone remember this proposed cheat?

Stop-and-wait Simulations - lookahead cheat

- Wait for everyone to send their updates, then send your update after taking into account their information
- Scenario: B makes a lethal shot at A
- A has a cheating agent that sends the decision to raise shields in time (because it looked-ahead)

Solution Version 1.0: Lockstep Protocol

- Prevents suppress-correct and lookahead cheats
- Anyone remember what the basic idea was?

Solution Version 1.0: Lockstep Protocol

- Basic Idea:
- Turn into 2-step process:
  - First, send hash of your decision
  - Second, once everyone has sent the hash, send the decisions
  - If hash(decisions) == hash_sent then good, else CHEATER!
- We've prevented the proposed cheats, but any Problems?

Solution Version 1.0: Lockstep Protocol

- Main problem is that we've just doubled the communication, and synchronization time since EVERYONE has to send the hash, before EVERYONE sends their decisions.
- This will slow us down - we're already battling time for reasonable user experience.
- Solution: Asynchronous version of Lockstep called "AS".

AS - Asynchronous Synchronization

- Basic idea is to restrict interaction of each player to an “aura” or SOI - sphere of influence
- If our player can’t be interacting with another player, then don’t bother synchronizing game play with them
- If we can interact (SOIs intersect) then lockstep with them

AS - Basic Algorithm Outline

- Compute local host's current state
- Send hash of it out
- Process hashes sent by remote hosts
- Foreach remote_host
  - Accept state of remote hosts (up to current time)
  - Compute potential influence (SOI's) intersections
  - IF no intersection of SOIs
    - THEN local host is not waiting for remote_host
  - ELSE IF hash(remote_host) accepted
    - THEN local host is not waiting for remote_host
  - ELSE local host is waiting for remote_host
- IF not waiting for any remote_host send local state

AS

- Prevents cheating same way lockstep does
- Prevents suppress-correct cheat because no dead reckoning
- Prevents lookahead cheat because we get hashes before we get real state (we don’t send our state, until we’ve received hashes from everyone else)

AS - Secret Possessions

- Can't really give knowledge of a "secret possession" to opposing players (assumption that they can see any information sent them)
- Instead we give them a "promise" - basically a hash or encrypted form of the real thing (if encrypted, the key is not known to opposing players)
- Player then signs the promise and gives it back - to prevent repudiation that the secret was promised.

AS - Secret Possessions

- Now, a player can have the promise and competing players won't know he has the secret possession (I guess everyone has a "promise" field or something)
- The game can be played
- Afterward, the promise can be verified against the "secret possession" to detect cheating (prevents opponents from cheating, only checks for local player cheating - not preventing it)

AS - MMO's

- AS like algorithm can be used for MMO's
- One cheating problem with MMO's is sending position information, or other information, to everyone - "it might affect player strategy" - this would alert of ambushes or the like.

- We assume the world is divided into regions
- How do we tell if a player is in the same region without exposing my region (which may warn of an ambush)?

- Proposed algorithm can do this.

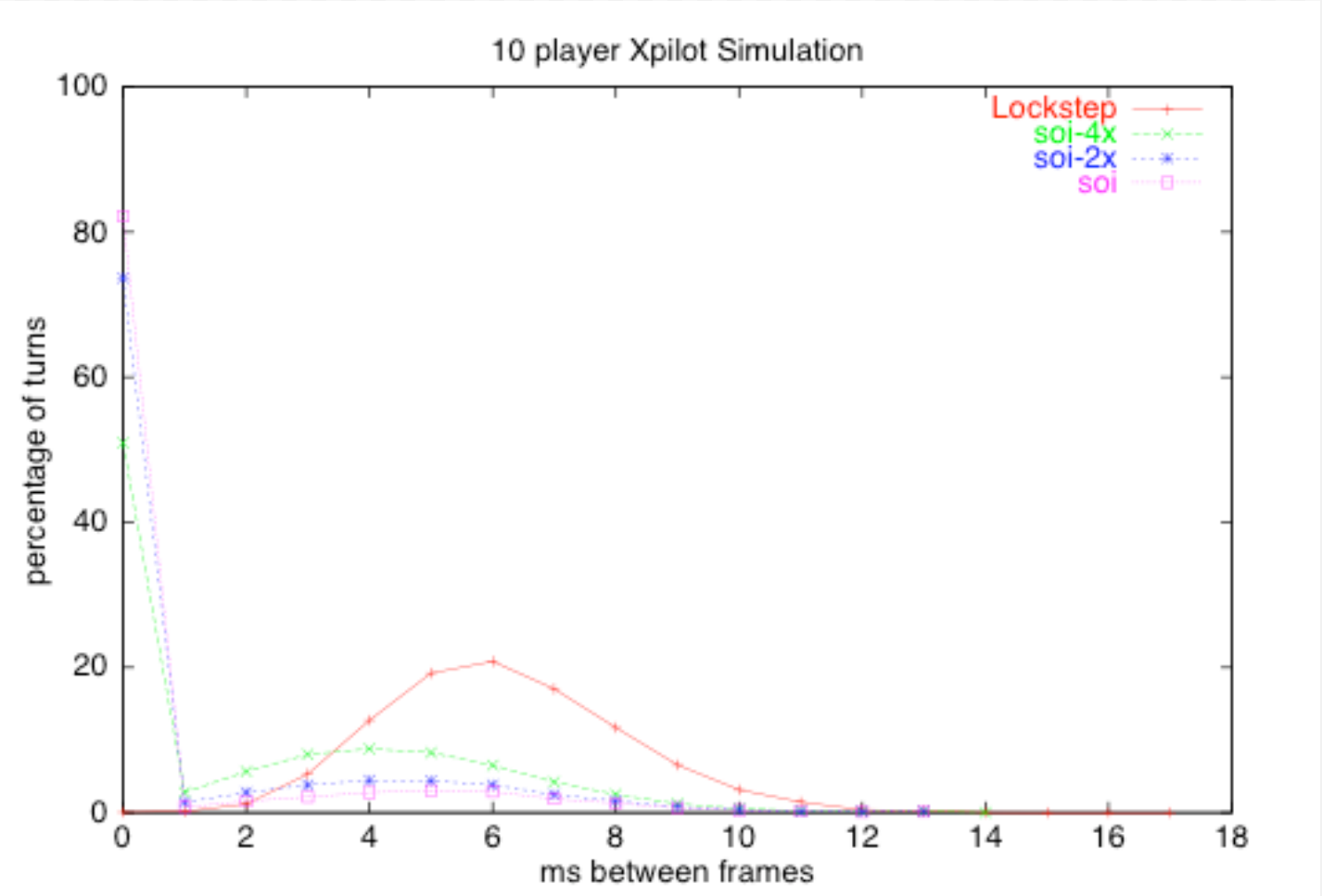
AS - MMO's - Hidden Positions

- Player A generates rand Ra and sends B h(Ra)
- Player B generates rand Rb and sends A h(Rb)
- Both compute R = Ra XOR Rb (? which means they exchange Ra and Rb as well right ?)
- A computes z = Ek(x+R), sends z to B
- B computers w = Ek'(y+R), sends w to A
- A computes w' = Ek(w), sends w' to B
- B computes z' = Ek'(z), sends z' to A
- A & B have z' & w', if z'==w' then x==y (same region)
- (because z'=Ek'(Ek(x+R)), w'=Ek(Ek'(y+R)) only difference can be from x and y)

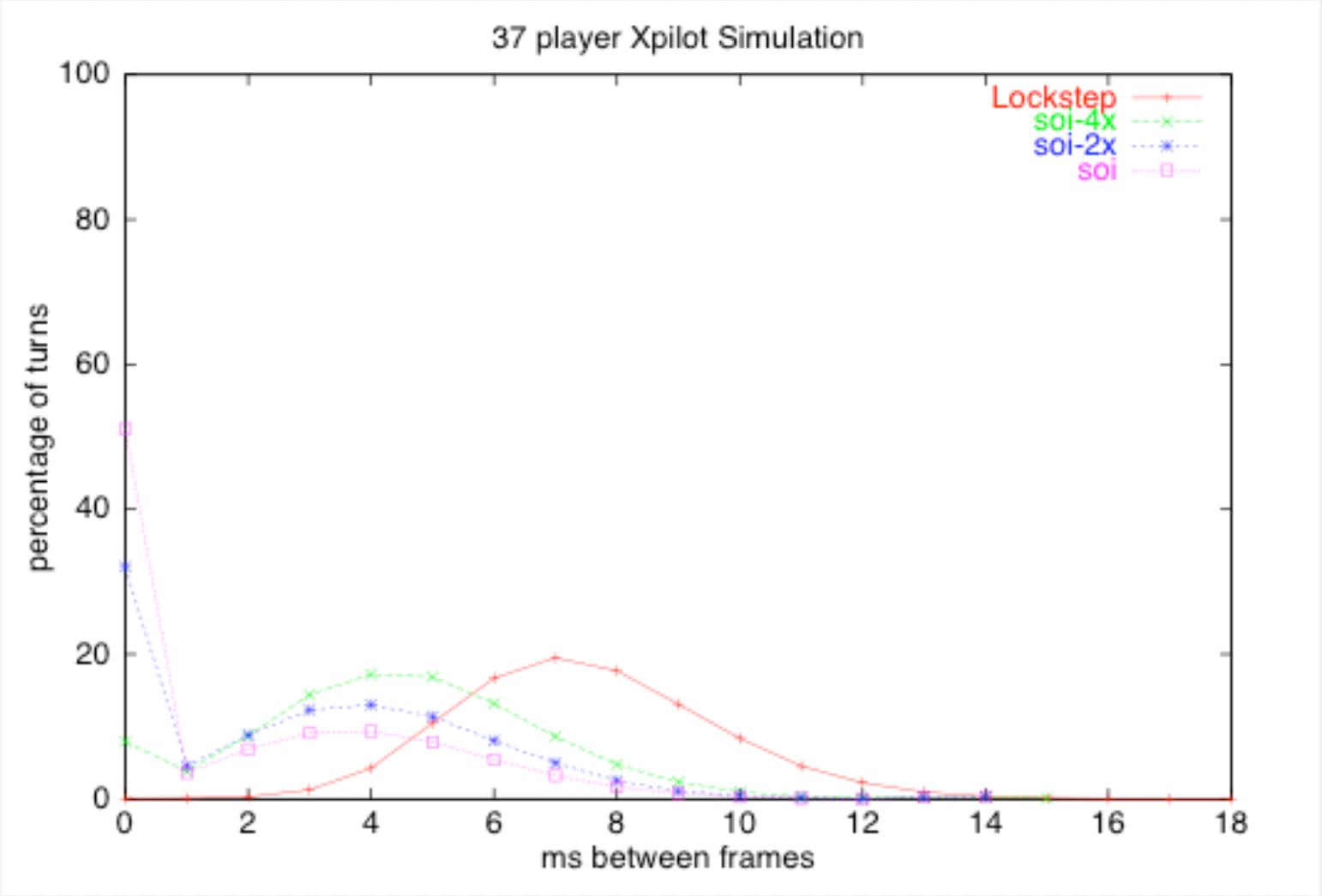
AS - Performance Comparison

- The reason we switched to AS was mostly due to performance
- Performance comparison used Xpilot game and 10,18,30, and 37 players
- Comparison was to see any improvement in frame rendering using AS over lockstep
- Results showed that AS always kept the ms between frames lower overall

AS - Performance Comparison



AS - Performance Comparison



Conclusions

- AS helps prevent two proposed cheats
- AS performs better than standard lock-step algorithm
- However, “it is clear dead reckoning will perform better…”(p.8)

Conclusions

- I appreciated their approach to make the game design resilient to hackers
- (traditionally just hack back at the hackers)

- But, they only really addressed a couple of possible cheats - it would have been nice to see more considered
- They claim it will scale for MMO's

- Comments?