Networking for Computer Games

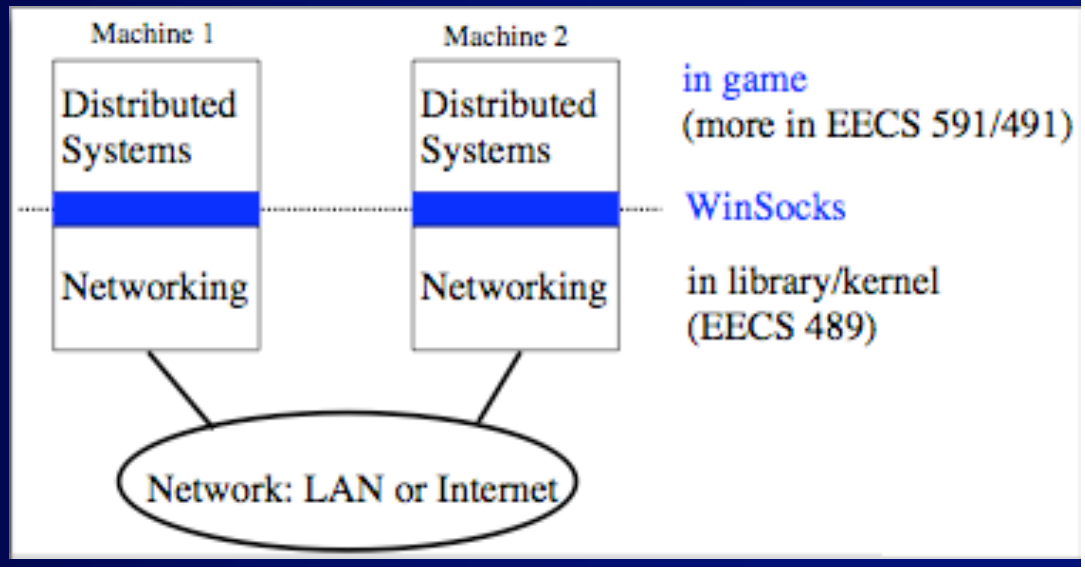
EECS 494

10/11/06 by Sugih Jamin

Networking in Games

In-game networking topics:

- networking topology: client-server vs. peer-to-peer
- computing model: distributed object vs. message passing
- which protocol to use? tcp, udp, reliable udp
- bandwidth limitation
- latency limitation
- consistency
- cheat proofing
- socket programming

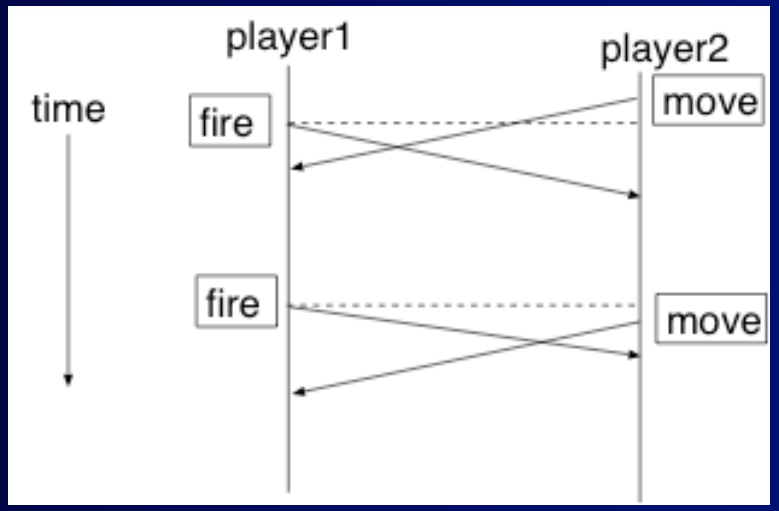


Consistency

Problem statement:

Case 1

Case 2



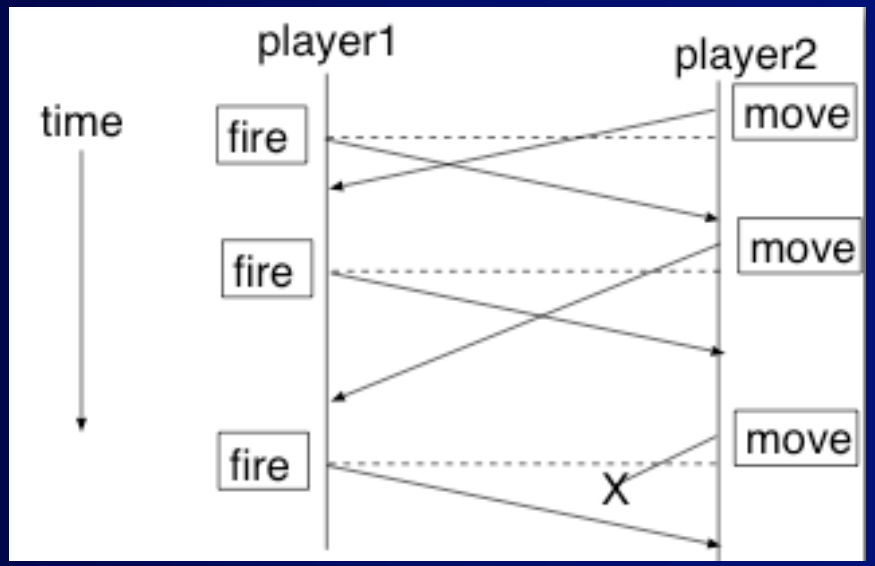
How do you differentiate the two cases, at both player1 and player2?

Synchronization

- Synchronization: order moves by their times of occurrence
- Assume globally synchronized clocks
- Out-of-synch worlds are *inconsistent*
- Small inconsistencies not corrected can lead to large compounded errors later on (deer not speared means one less villager means slower barrack build, etc.)

When to Render a Move?

How long do you have to wait for the other players' moves before rendering your world?

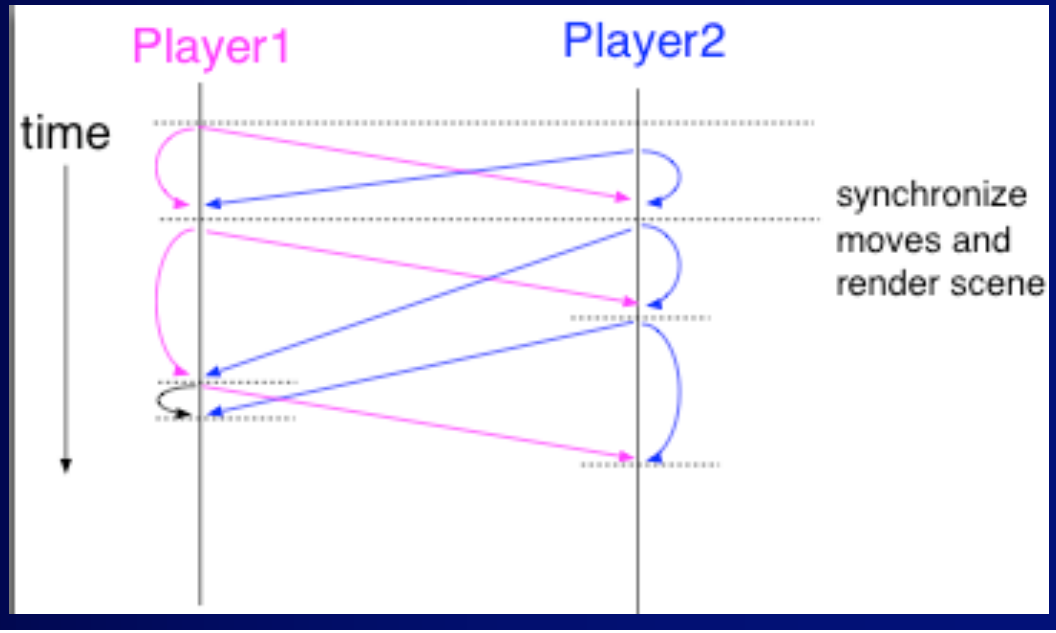


Lock-step Protocol

Algorithm: Each player receives all other players' moves before rendering next frame

Problems:

- long Internet latency
- variable latencies
- game speed determined by the slowest player



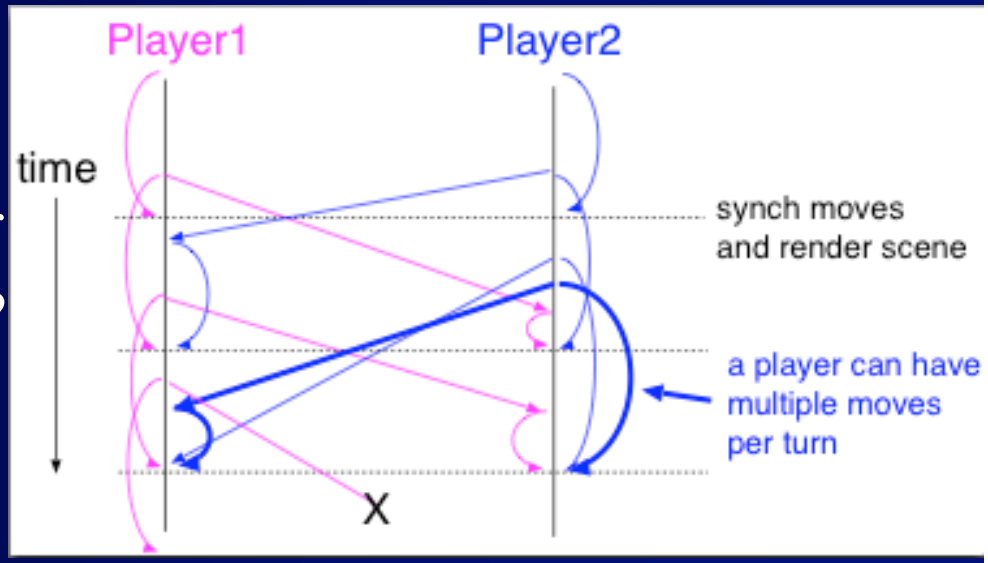
Bucket Synchronization

Algorithm:

- buffer both local and remote moves
- play them in the future
- each bucket is a turn, say for about 200 ms
- bucket size can be adapted to measured rtt

Problems:

- game speed (bucket size) determined by slowest player
- what if a move is lost or late?



Pessimistic Consistency

Every player must see the EXACT same world

AoE/AoK/AoM:

- each player simulates its own copy of the world
- all the worlds must be in sync.
- uses bucket synchronization
- each player sends moves to all other players
- dropped packets retransmitted
- a designated host collect measured rtts from all players and set future bucket sizes

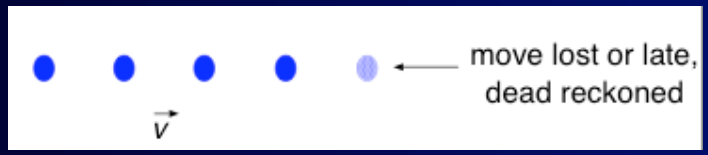
Problems:

- variable latencies
- speed determined by the slowest player

Dead Reckoning

Dead reckoning, a.k.a. client-side prediction

- extrapolate next move based on prior moves
- compute the velocity and acceleration of objects to dead reckon
- players can help by sending this info along
- obviously, only works if velocity and acceleration haven't changed



Roll-back

In case of inconsistency:

- server always have authoritative view
- when clients correct inconsistent views, players may experience ``warping''
- can players' decisions be dead reckoned? (see http://spectrum.ieee.org/sep06/4424)

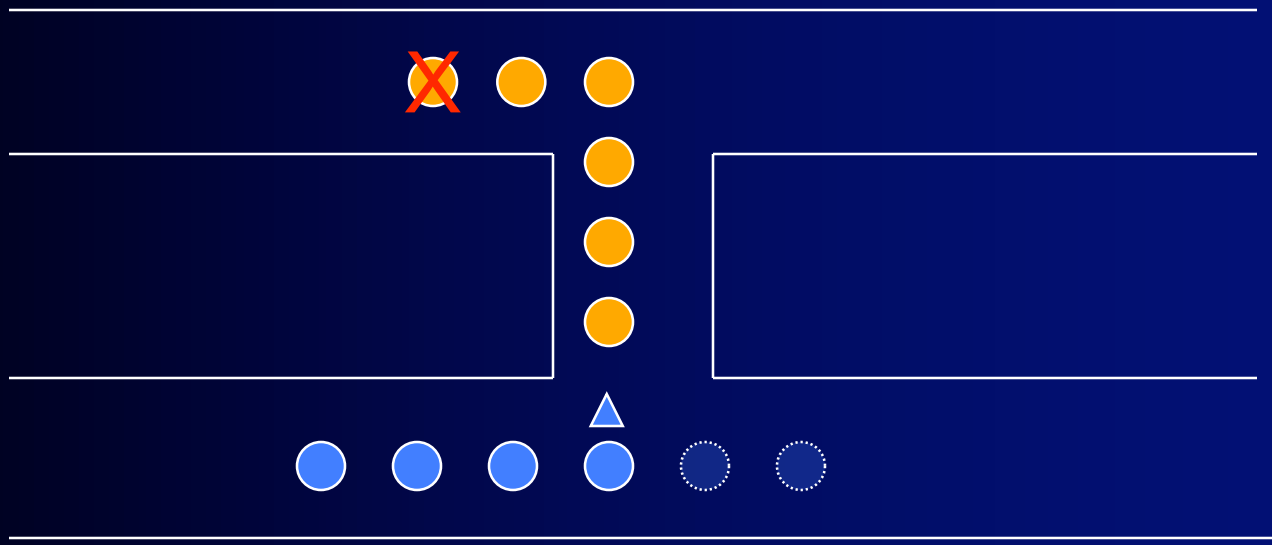
Optimistic Consistency with Roll-back

Observation: dead reckoning doesn't have to be limited to lost packets!

Half-Life:

- each client plays back its own moves immediately and send the moves to server
- each client also dead reckons the other players' moves
- server computes world and sends its authoritative version to all clients
- clients reconcile dead reckoned world with server's version
- can result in some jerkiness and perception of "shooting around corner"
- only need to synchronize important events, but must be careful that dead reckoning error doesn't get compounded over time

Shooting around Corner

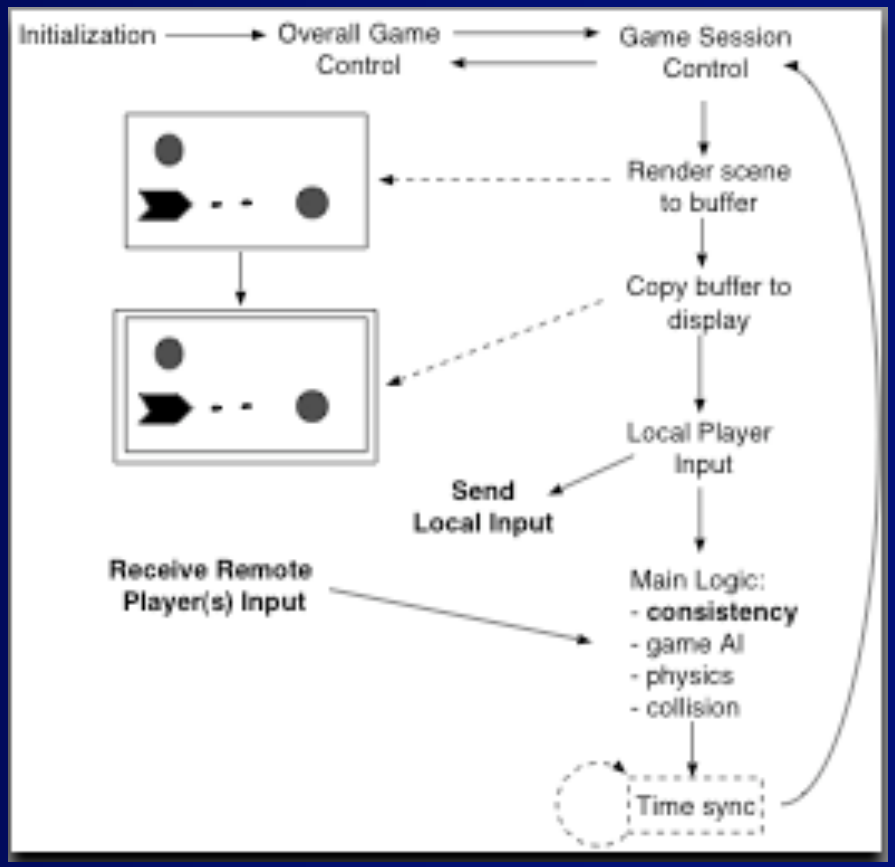


Consistency: Correctness

For consistency ALL user input MUST pass through the synchronization module

Be careful with random number generators. Isolate the one used for game-state updating from other uses (ambient noise etc.)

Design for multiplayer from the start. Single-player becomes a special case of single-client multiplayer game



Consistency: Smoothness

For smoother playback, decouple bucket size from frame rate (even AoE does this)

Immediately render local moves

Modify game design to allow for latency and loss, e.g.,

- make players wait for elevator
- teleportation takes time
- require multiple hits per kill
- let bullet/missile have flying time
- build in inertia, don't allow sudden change in facing

Reducing Consistency Check

Do *area-of-interest* management (a.k.a. relevance filtering):

- *aura*: how far you can be sensed (cloaked ships have ε aura)
- *nimbus*: how far you can sense (use quantum-sensor to detect cloaked ships)

Aura and nimbus are defined for a given set of ``technology'' (e.g., cloaking device, quantum sensor, etc.)

Perform consistency check only when *B* is within *A*'s nimbus and *A* is within *B*'s aura

Cheating

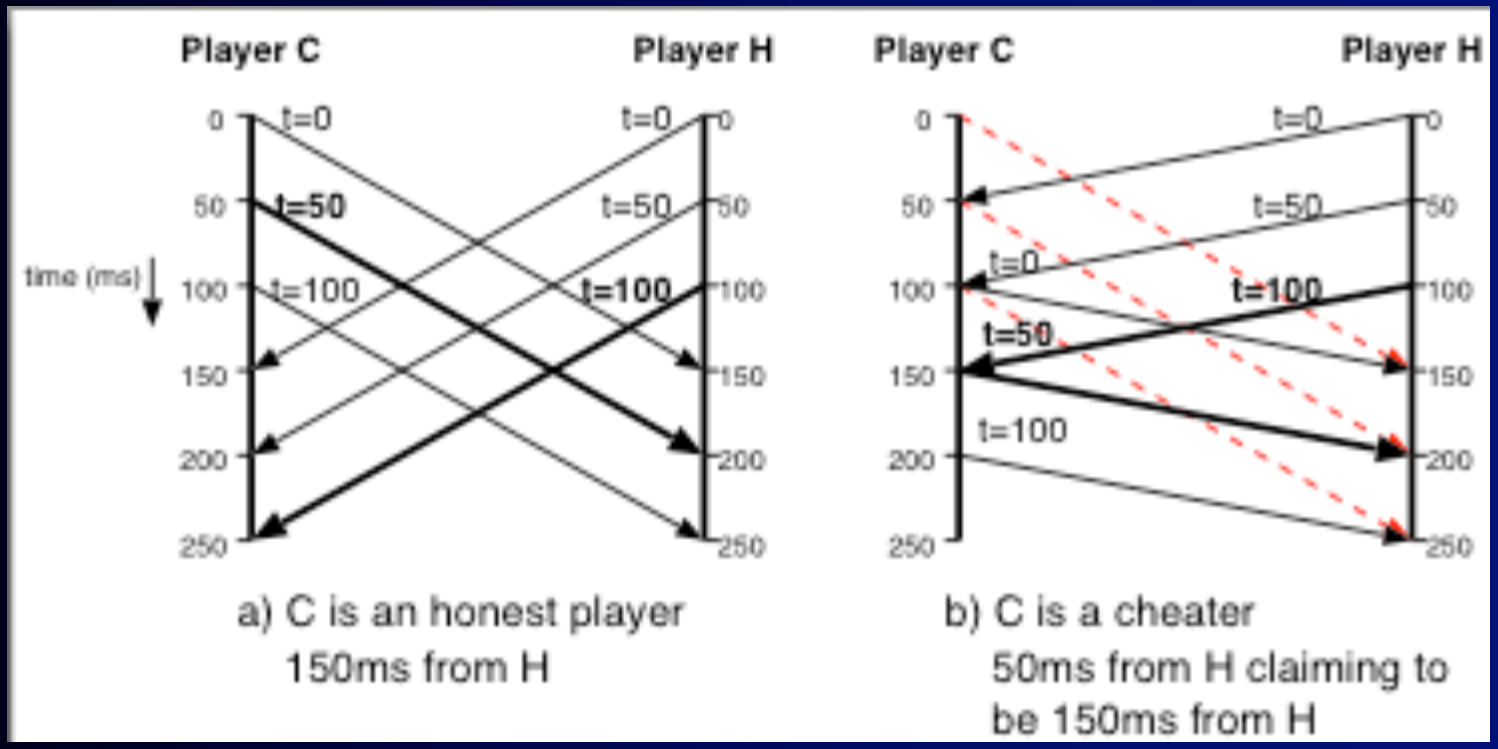
AoE doesn't need cheat-proofing because each player simulates each move in lock step: all moves are simulated, not just collisions

Half-Life synchronizes only collisions, higher probability for cheating

Cheats (more at megagames.com):

- superhuman cheat: auto-aim, auto-position
- game-state editing: boost player's profile
- rule bending: see/walk through walls
- sixth-sense cheat
- lookahead cheat: claim to be behind slow link
- suppress-correct cheat: exploit dead-reckoning, claim moves were lost, then ``reconstruct'' advantageous moves based on others' moves

Lookahead Cheat

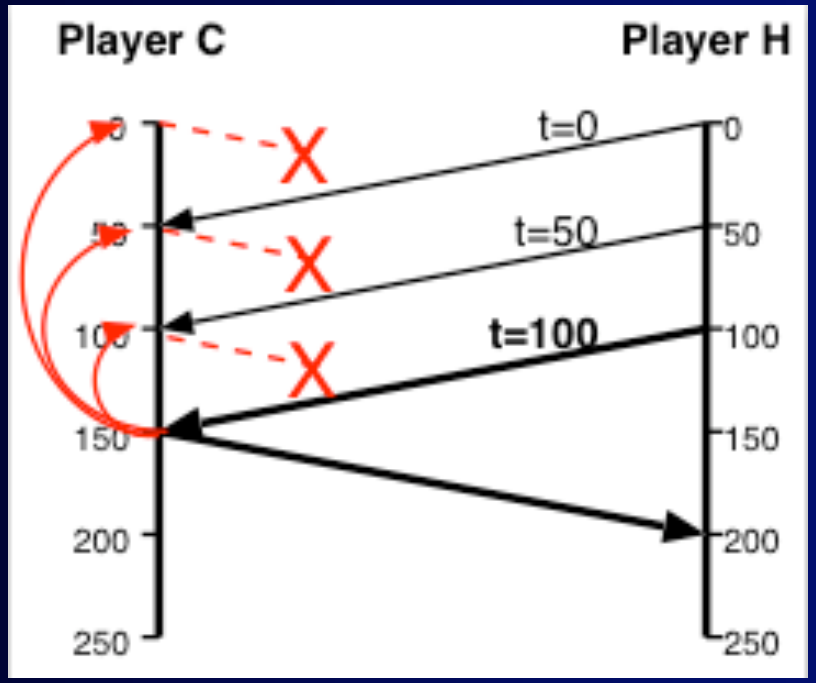


a) C is an honest player 150ms from H

b) C is a cheater 50ms from H claiming to be 150ms from H

Suppress Correct Cheat

At time 150, *C* sends out a move consistent with fake moves at time 0, 50, 100 that were actually computed upon receiving packets from *H*



Distributed Computing Model

Two common models:

- Distributed Objects
- Message Passing

Both implemented as abstractions over the socket APIs:

- Distributed Objects: object update library sends/receives object states using socket APIs
- Message Passing: player inputs sent/received using sockets

Socket Programming

What is a socket?

How to use socket for client-server computing?