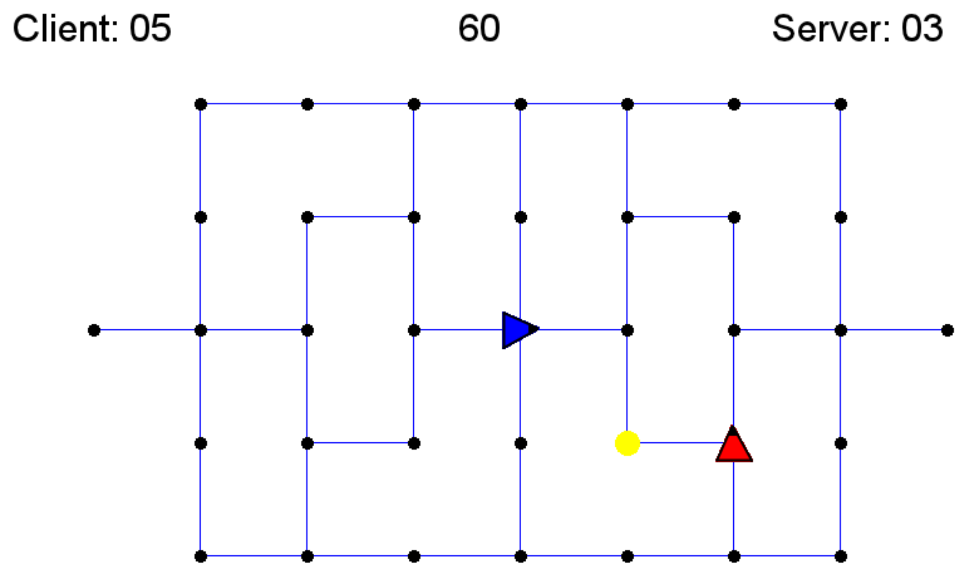
Latency & State Consistency in Networked Real-Time Games



Christopher D. Canfield

CS 535 Summer 2013

Introduction

- In networked real-time games there is an inherent tradeoff between latency reduction techniques and the consistency of the distributed state [12], [14], [20]
    - True of any distributed virtual environment

- By studying the impacts on users of network latency and distributed state inconsistency, stakeholders in the games industry can make informed decisions on which tradeoffs are appropriate in which cases

- Additional (non-game) uses for these techniques: distributed military simulations (multiple algorithms listed in later slides were originally used for that), virtual training simulators, etc

* See project paper for references

Impacts on User Satisfaction & Performance from Network Latency & State (In)Consistency

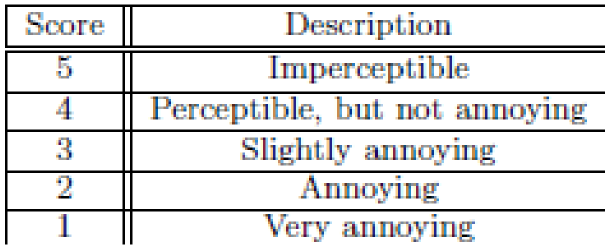
- Delays under 50 milliseconds do not impact player performance [16]

- Delays over 50 milliseconds but under 100 milliseconds begin to have a slight impact player performance, but are rarely noticed by the player [4], [9], [12], [16], [19]

- Delays greater than 100 milliseconds are noticeable in fast-paced real-time games with a first person perspective, such as racing games and games in the first-person shooter genre. Player performance begins to fall at this level. [1], [2], [16]

- Players will migrate away from FPS servers that result in a latency greater than 150-180 milliseconds [1]

Impacts on User Satisfaction & Performance from Network Latency & State (In)Consistency



Score	Description
5	Imperceptible
4	Perceptible, but not annoying
3	Slightly annoying
2	Annoying
1	Very annoying

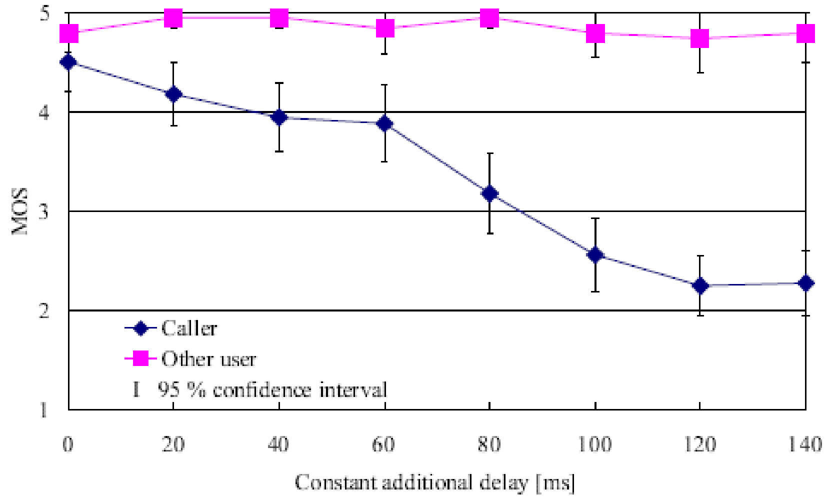


Figure 4: MOS versus constant additional delay in the one-way case.

In a networked rock-paper-scissors experiment using real-time video and audio, Hashimoto and Ishibashi found that a user's mean opinion score, declined from 4 (“Perceptible, but not annoying”) at 60 milliseconds delay to approximately 2 (“Annoying”) at 120 milliseconds [9]

Impacts on User Satisfaction & Performance from Network Latency & State (In)Consistency

- Chen, Huang and Lei found that absolute delay due to the network mattered less to players than delay jitter and packet loss [3]

- "if players quit because they are frustrated by unfavorable network conditions, on the average, 10 percent of their dissatisfaction is caused by network latency, 20 percent by network delay jitter, 40 percent by client packet loss, and 30 percent by server packet loss."

- Whether a player could leave the server could be accurately predicted based on current network conditions and length of time player had been on the server

- Players are "sticky": the longer they are in a game, the more likely they are to stay, regardless of network conditions

Impacts on User Satisfaction & Performance from Network Latency & State (In)Consistency

- Their recommendation: Allocate more resources to players when they first connect, and then reduce the amount of resources allocated to them over time

- Caveat: study focused on a game in the MMORPG genre, which has different player tolerance characteristics than games in other real-time genres, particularly first-person shooters

Latency Reduction & State Consistency Methods

Pessimistic Methods, or “Consistency is Key”

- Ensure consistency at the cost of additional latency
- In general, they will not allow the game/simulation to move forward until all players are in sync

Latency Reduction & State Consistency Methods

Pessimistic Methods: Lockstep Synchronization

- Requires game state from all users to be transmitted and confirmed before the simulation clock can be incremented [5]

- Ensures complete consistency among the distributed users, at the cost of potentially severe delay

- Used (or was used) in certain distributed military simulations, but is not appropriate for consumer games, since there is no way to guarantee the speed

Latency Reduction & State Consistency Methods

Pessimistic Methods: Bucket Synchronization

- Time is divided into fixed length buckets

- Update messages are allocated to different time buckets depending on when they were issued by senders (requires clocks to be synchronized)

- To calculate the updated global state, a participant processes all events in the current bucket

- By introducing a synchronization delay (inherent in dividing time into buckets), the impacts of network latency and variability is reduced [5], [18]

Latency Reduction & State Consistency Methods

Pessimistic Methods: Bucket Synchronization

- Bucket Synchronization can be modified to be optimistic:
  - After a set delay, any missing updates are *dead reckoned* (explained in the Optimistic Methods section), which creates inconsistencies

Latency Reduction & State Consistency Methods

Pessimistic Methods: Artery

- Created in response to the high overhead of the IEEE 1278 Distributed Interactive Simulation (DIS) and BBN SIMNET military simulation protocols

- Groups nodes (players + simulated entities) into consistency groups based on spatial characteristics

- The state among nodes in a group is always consistent, but different groups can be inconsistent

- Uses *dead reckoning* and message aggregation to further reduce bandwidth requirements

Latency Reduction & State Consistency Methods

Optimistic Methods, or “Latency Reduction First”

- Execute events before knowing for certain that no earlier events will arrive

- Reduces perceived latency, by ensuring that the simulation continues updating, at the cost of inconsistency between different copies of the game environment

Latency Reduction & State Consistency Methods

Optimistic Methods: Dead Reckoning

- The next state of each game entity is predicted by all peers by using locally available information, such as position and velocity

- A controlling application for each simulated game entity determines if the calculations performed by the distributed peers have deviated too far from the actual state

    - If so, the controlling application sends an update message

- Reduces perceived latency and required bandwidth (fewer messages)

Latency Reduction & State Consistency Methods

Optimistic Methods: Dead Reckoning

- Drawback: loss of state consistency can result in odd situations when state is corrected

- Example:
  - Suppose that a car driven by Player A in a racing simulator is travelling in a straight line
  - On Player B’s machine, a dead reckoning implementation may predict that the car will continue moving forward, and update the car’s position as such
  - If Player A actually made a turn, the controlling entity will eventually note the inconsistency and send an update packet
  - The car will appear to “warp” to the correct position on Player B’s machine

Latency Reduction & State Consistency Methods

Optimistic Methods: Timewarp (aka Time Warp)

- Extends dead reckoning by enabling rollback to a previous, confirmed state when an inconsistency is detected

- Brief overview:
    - Each distributed application periodically saves a snapshot of all simulated entities
    - Each application keeps a list of events that is has received
    - New events are inserted into the event list as they arrive
    - When an inconsistency is detected, a rollback followed by a state re-computation occurs, as described above

Latency Reduction & State Consistency Methods

Optimistic Methods: Timewarp (aka Time Warp)

- Anti-messages are sent as part of the rollback to cancel events which have become invalid

- Problem: these anti-messages can generate additional anti-messages from invalidated events, which in turn can generate additional anti-messages, etc

Latency Reduction & State Consistency Methods

Optimistic Methods: Local Lag

- Delays local events to approximately match the latency of the network

- If the delay value required to match the network latency is too high, local lag can be combined with dead reckoning or timewarp

- Local lag is essentially a buffering mechanism, helping to eliminate the effects of high jitter

- If a high enough value is selected, most inconsistency can be eliminated, but at the cost of causing noticeable lag (essentially a pessimistic method at that point)

Latency Reduction & State Consistency Methods

Optimistic Methods: Trailing State Synchronization

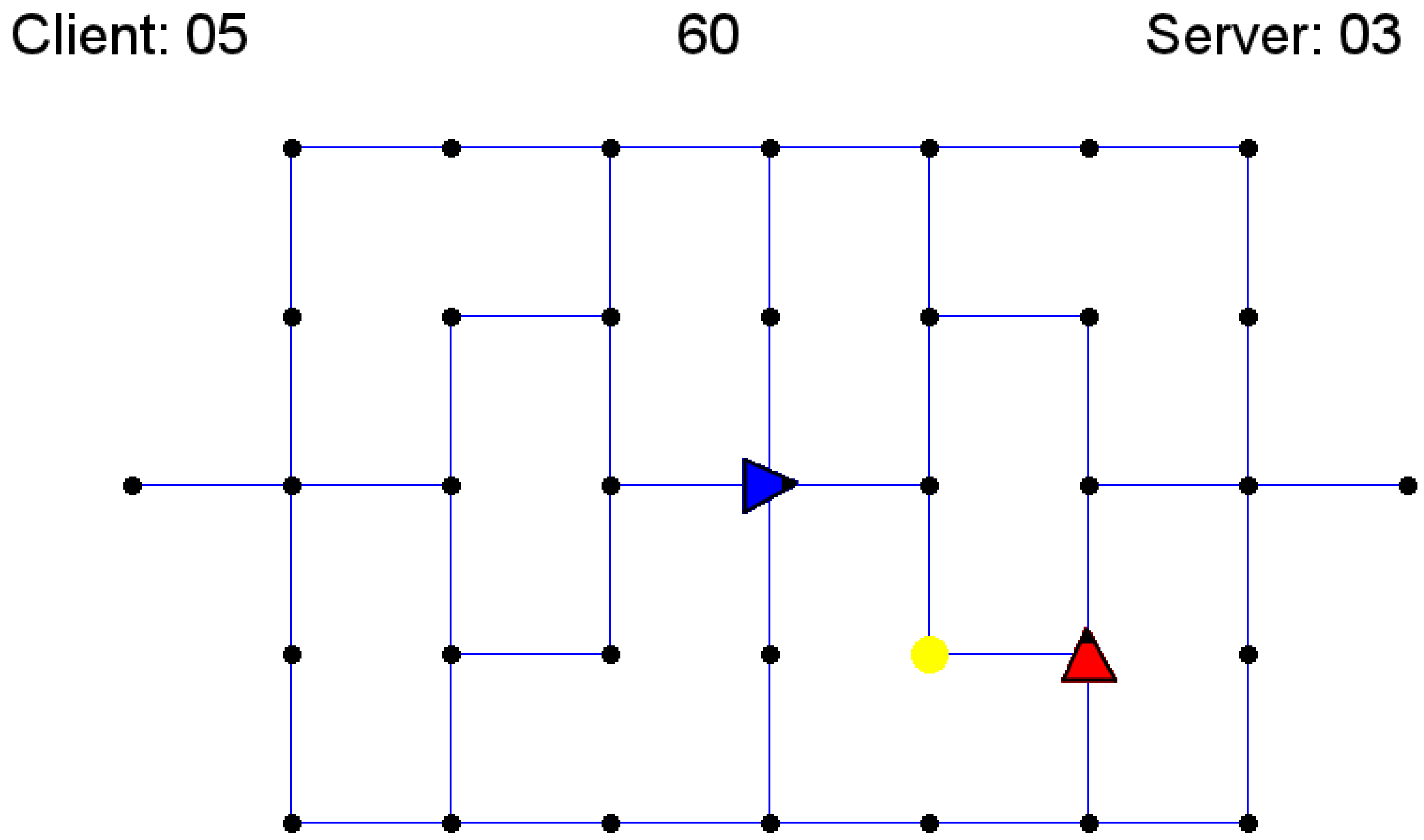
- Runs multiple copies of the game state, each at a slight lag to the next
- Each trailing state executes all commands, but on a lag to the foremost state
- If the system determines that its current state is invalid, it rolls back to a previous state
  - The event that caused the invalidated state is placed into the event queue
  - The event queue is replayed up to the current time to generate the correct state
- Similar to Timewarp, but lower cost in memory (no per-event state saves) and bandwidth (no anti-messages)

Latency Reduction & State Consistency Methods

Optimistic Methods: Rendezvous

- Designed for "high latency collaborative systems, including mobile multiplayer games." [2]
- Each game node operates asynchronously from all others.
- *Rendezvous arbitrators* ensure that each asynchronous game view does not become too disconnected from the others
- They do this by generating *target states*, which are points at which the game views can become consistent
- It is then up to the *adaptation rules* to push the local view toward the target state, by controlling simulated entities and subtly bending the rules of the game universe.

Experiment: Latency Testing with Triangle Wars



Experiment: Latency Testing with Triangle Wars

- Networked real-time game with network delay simulator
- Implemented after reading the literature described in the previous slides

- Goals:
  - Perform preliminary testing on player fairness perceptions at varying network latency levels
  - Learn or partially learn how network delay and other impacts can be simulated for testing purposes
  - Create the initial implementation for a system that can be extended in the future to include various algorithms described in the previous slides
  - Implement an application that interacts directly with the Transport Layer, without using an existing intermediary Application Layer protocol, enabling the author to put into practice a small part of what we have learned in class

Experiment: Latency Testing with Triangle Wars

- Written in C++ and uses the cross-platform Simple & Fast Media Library (SFML)

- UDP was used at the Transport Layer, since this was true of all games described in the literature except for MMORPGs, which frequently use TCP

- The architecture is client-server, with certain peer-to-peer modifications
    - For example, while the server performs the goal check for all players (i.e., it checks to see if the player has picked up the yellow goal circle, which increments the player's points by one), the client player performs its own movement validity checks

Experiment: Latency Testing with Triangle Wars

- Application-level packet headers consist of a one-byte packet type identifier.

- The data payload ranges from one byte for the smallest packet type to 16 bytes for the largest type (the server-to-client update packet).

- Network delay for the client can be simulated by delaying the send time of the packet to the client (specified at the start of a match)

- A fast network link with a round trip time of less than 1 millisecond was used to ensure that any delays encountered were from the simulated delay

Experiment: Latency Testing with Triangle Wars

- Simple gameplay:
  - Two opposing triangles start at opposite sides of an arena
  - At match start, a yellow goal circle is randomly placed in one of the center waypoints.
    - This is generated by the server player's machine, which gives the server player an advantage if network latency is high, since the client won't know where the goal is. This was by design.
  - Players receive one point per goal that they pick up

Experiment: Latency Testing with Triangle Wars

- Simple gameplay:
  - Two opposing triangles start at opposite sides of an arena
  - At match start, a yellow goal circle is randomly placed in one of the center waypoints.
    - This is generated by the server player's machine, which gives the server player an advantage if network latency is high, since the client won't know where the goal is. This was by design.
  - Players receive one point per goal that they pick up

Experiment: Latency Testing with Triangle Wars

- Simple gameplay:
  - Two opposing triangles start at opposite sides of an arena
  - At match start, a yellow goal circle is randomly placed in one of the center waypoints.
    - This is generated by the server player's machine, which gives the server player an advantage if network latency is high, since the client won't know where the goal is. This was by design.
  - Players receive one point per goal that they pick up

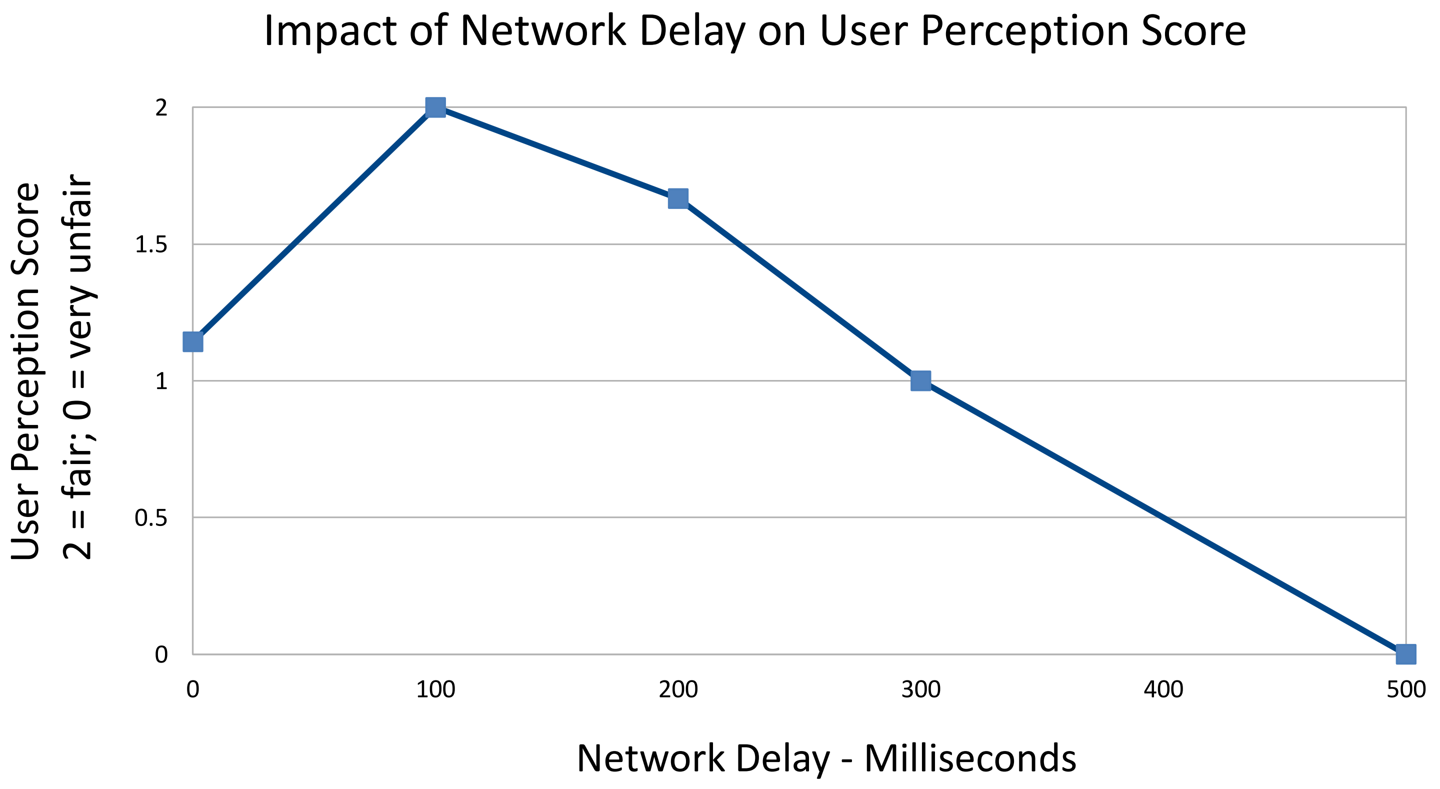
Experiment: Latency Testing with Triangle Wars

Methodology

- Training rounds to familiarize player
- User given a sheet of paper numbered 1 to 20
- Instructed to label each game as 2 (fair), 1 (slightly unfair) or 0 (very unfair)
- Each game used a randomly assigned additional packet delay, which was simulated by the game, of 0, 100, 200, 300, and over 500 milliseconds
- (actual network RTT according to Windows ping utility was 0 milliseconds)

Experiment: Latency Testing with Triangle Wars

Results



Experiment: Latency Testing with Triangle Wars

Results

- Results consistent with literature

- However, findings are only preliminary, and are not scientifically valid, because of the small sample size

Experiment: Latency Testing with Triangle Wars

Future Research

- Re-run tests with larger sample size, and then perform proper statistical analysis on results
- Add subset of algorithms listed in previous slides, and re-run tests
- Add ability to simulate network jitter and packet loss, in addition to the already-implemented constant delay
- Perform the perception tests with additional simultaneous players
    - Would more players distract the user from latency?

Thank you