QoS and QoE

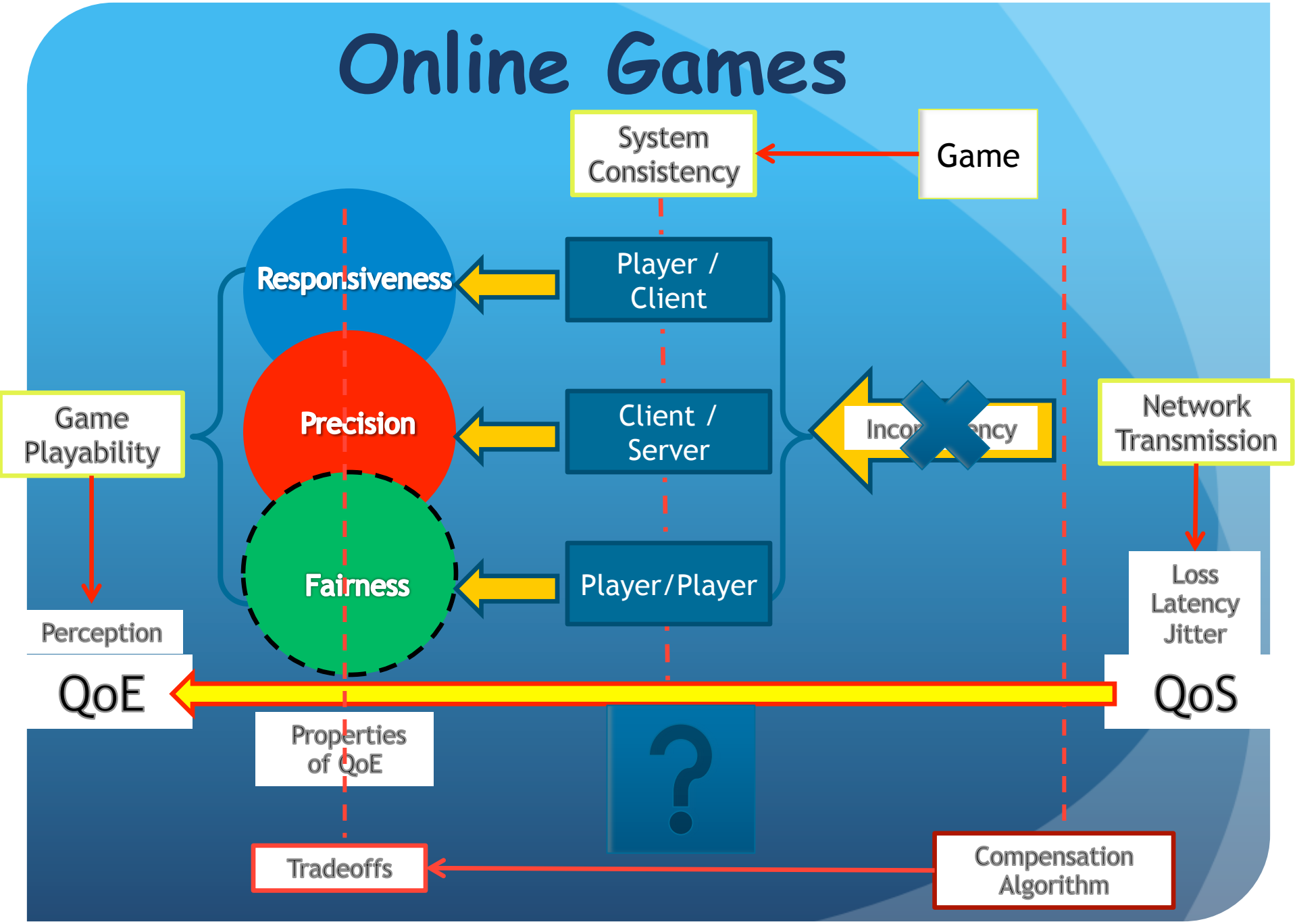
Magda El Zarki
Professor of CS
UC Irvine
Email: elzarki@uci.edu
http://www.ics.uci.edu/~magda

Network Impairments: Packet Loss, Delay and Jitter

- Network Measurements:
  - Performance Parameters - e.g., average end to end delay, maximum jitter, % packet loss
  - Service Guarantees – Quality of Service (QoS) that can be expected for a particular application.
  - Service Contracts – Ensure that e.g., 10% of time the QoS is met on all traffic flows of an application or service traversing a network
- How to capture their impact on end users and applications? Quality of Experience
  - User experience
  - Impact on game playability

QoS vs QoE

- QoS – Quality of Service:
  - network characteristics/behavior
  - Performance guarantees given by network provider based on measurements
- QoE – Quality of Experience:
  - impact of network behavior on end user
    - some imperfections may go unnoticed
    - some imperfections may render application useless
  - not captured by network measurements
    - a 5% packet loss could be invisible if it affects background
    - a missed target due to a 100ms delay can affect game outcome


Online Games
System Consistency
Game
Responsiveness
Player / Client
Precision
Client / Server
Fairness
Player/Player
Game Playability
Network Transmission
Loss
Latency
Jitter
Perception
QoE
QoS
Properties of QoE
?
Tradeoffs
Compensation Algorithm

Latency Compensation

Description, Causes and Corrections

Outline

- Introduction
- Need for Latency Compensation
- Techniques
  - Prediction
  - Time Manipulation
  - Visual Tricks
- Latency Compensation and Cheating

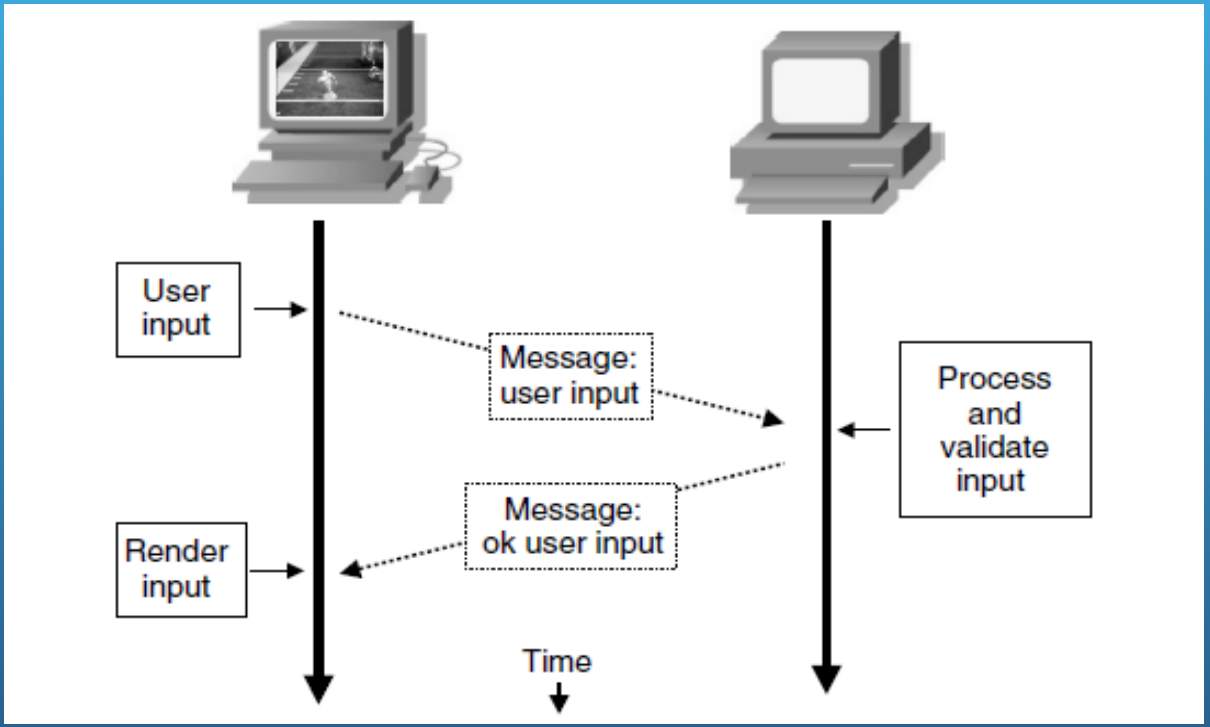
Client Server Architecture

- There is a single dedicated server which performs all the intelligent functions and controls all operations.
- All other client machines connected to the server are dumb clients which only provide visual aid to the users and collects user input.
- All user requests (inputs) are sent from the *dumb* client to the server which carries out the request and sends the result back to the client.

Latency in Network Games

- Most games today run on the client server architecture with a single server which handles the game logic and every request made by a player in the game is processed at the server.
- Thus, when a request is made by a player, it travels from the client to the server and back to client, and this transmission introduces possible latency in the game.

Schematic View of Latency



What is Latency Compensation?

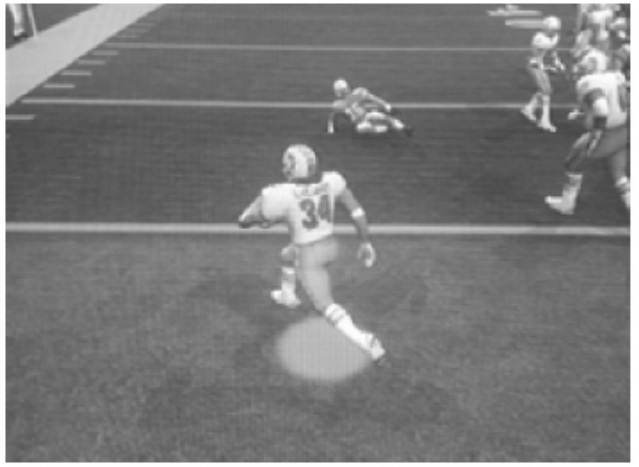
- Latency Compensation is the method of minimizing and hiding latency of the network through various techniques, so that the system appears very responsive and impressive.

Need for Latency Compensation: Example 1

Illustration of the effects of latency on running (Madden NFL 2003).

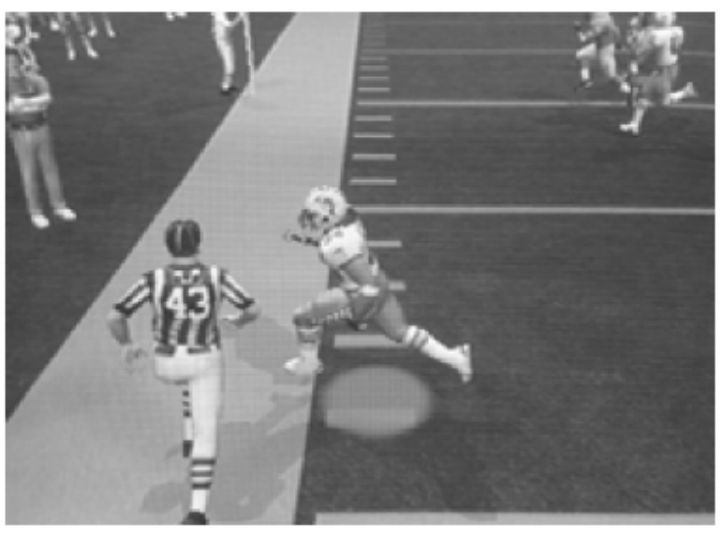
1. User is pressing left and the player moves left.
2. User is pressing up, but player continues left because of latency.
3. Running back goes out of bounds!

1.



2.

3.

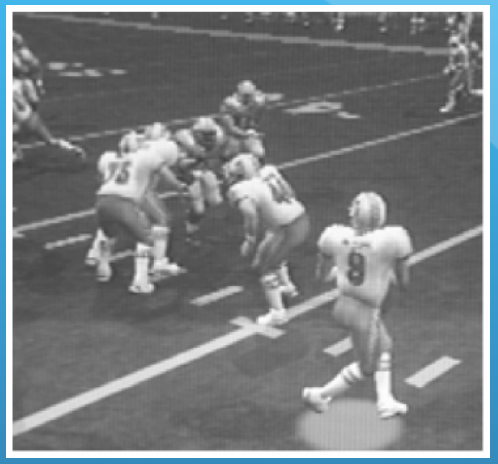


Need for Latency Compensation: Example 2

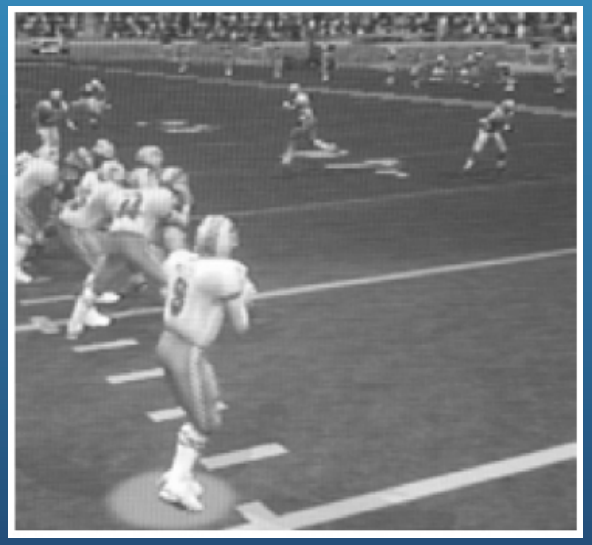
- Illustration of the effects of latency on running (Madden NFL 2003).

1. User is pressing throw, but throw is not processed yet because of latency.
2. Throw starts processing here because of latency
3. Defender intercepts throw!

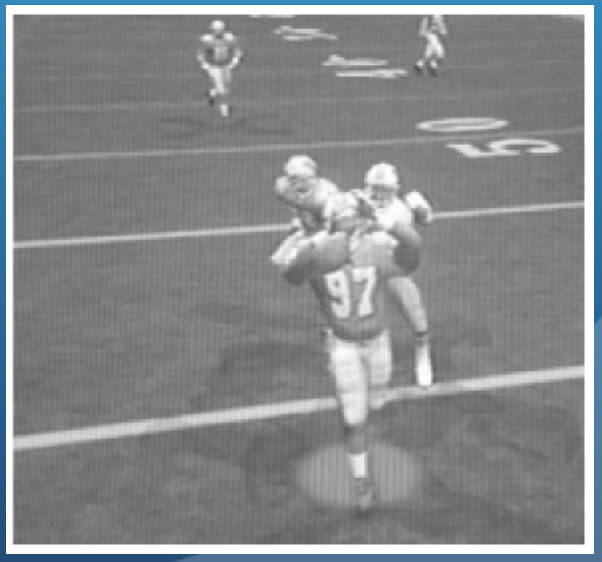
1.



2.



3.



Techniques for Latency Compensation

- There are various methods that can be applied to compensate for latency or mute its effect:
  - Prediction
    - Player
    - Opponent
  - Time Manipulation
    - Time Delay
    - Time Warp
    - Data Compression
  - Visual Tricks

Prediction

- Two classes:
  - Player prediction: The game client responds to the user immediately without any delay (i.e. waiting for server to acknowledge action) -> the local system renders the client's actions immediately
  - Opponent prediction: The client predicts the behaviours of the opponent before the server actually sends them to the client

Player Prediction

- The client takes input from the user (player)
- The client predicts the server response only for the player's actions
- The player’s actions are not subjected to the round trip delay (to the server and back), thus removing any network latency
- The game appears very responsive -> like a non-networked game

Prediction Algorithm

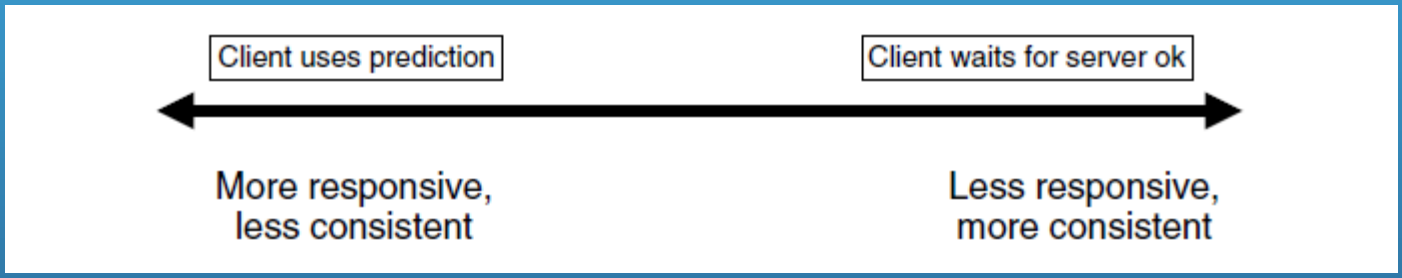
- Sample user input
- Pack up data and send to server
- Determine visible objects and game state
- Render scene based on local actions
- Receive updates from server and unpack
- Correct for any discrepancies
- Repeat.

Drawbacks:

- If there is a discrepancy between the actual server game response and the client side prediction, the client needs to make the adjustment in the game state
- These adjustments could be abrupt and could cause jitter and deteriorate the consistency of the game. A client could use ***Interpolation***, between the rendered local world and the server update showing the position of units at a later time. But instead of immediately rendering the latest update the client *interpolates* the world at intermediate states, allowing the local world state to progress smoothly to the server world state.
- There is a constant trade-off between game responsiveness and game consistency

Drawbacks

Responsiveness and consistency have a constant tradeoff.



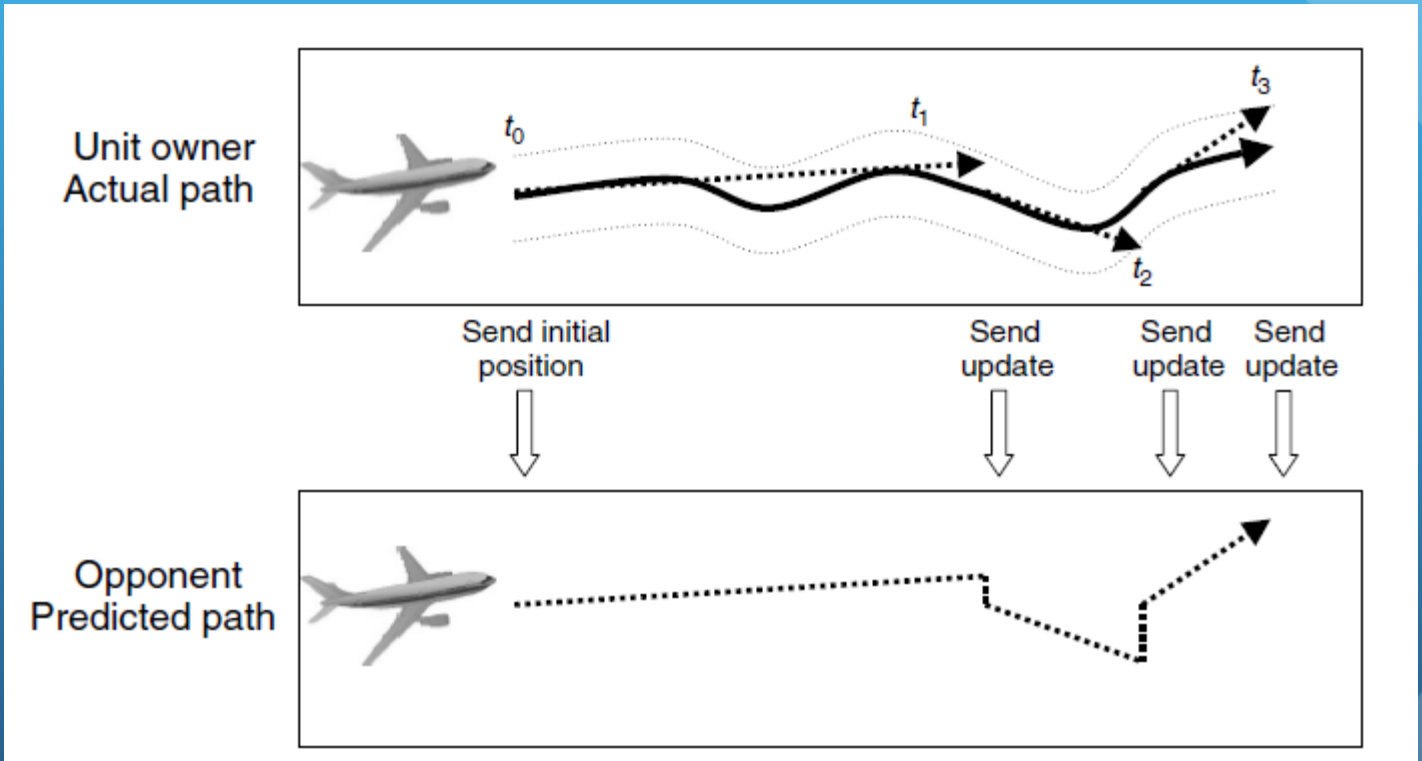
Opponent Prediction

- In this technique the location and movement of an opponent's unit is predicted by the client
- It starts on the basis of the unit's last known position, and computes the current predicted position based on the unit's speed and direction of movement.
- The predicted position of the opponent's unit is used until the opponent sends an update for the new position within a predetermined threshold/error limit.

Opponent Prediction Cont'd

- The update is sent when the unit owner determines that the other players are not able to accurately predict the position within the predetermined threshold
- The update sent by the unit owner contains the correct position, velocity and orientation of the unit
- *Predetermined threshold* refers to the acceptable range within the original position of the opponent, where the opponent's unit can be placed for assuming correct prediction.

Opponent Prediction Cont'd



Opponent Prediction Cont'd

- The picture shows the view by the owner of the unit.
- The solid line in the middle shows the actual path as the player controlling the unit would see it.
- The two thinner, dashed lines that run parallel to the middle line represent the threshold for the opponent predictions.
- The thicker dashed lines represent the unit owner's record of the opponent's prediction.
- If the unit owner's predicted location of the airplane goes outside this threshold, the unit owner sends a message to all opponents with an update on the new position and direction.

Opponent Prediction Cont'd

- The initial position and direction is sent at time t_0, and subsequent updates are sent at t_1, t_2, and t_3.
- The bottom picture depicts the view that the opponents would see.
- The opponent uses the last known position and direction to predict the unit location until an update is received, whereupon the new position and direction are used.

Trade-offs

- It requires that each client run an algorithm to extrapolate the location of each unit for each frame rendered
- Smaller values for the update threshold can provide more fidelity in opponent's prediction at the cost of requiring more frequent updates. This requires higher bandwidth and processing overhead.
- On the other hand, larger values of the update threshold provide decreased fidelity but requires a lower update rate, therefore less bandwidth and lower processing overhead.
- This tradeoff depends on the game, the network and the client processor and the players' preference.
- Fairness – players further away get updates to locations later than others and so are at a disadvantage with seeing a wrong "game view". Solution: send clients further away more location updates.
- Errors can result in game disruptions. Some games use a game controlled avatar to adjust position of the object, e.g., position of a soccer ball – computer controlled player can be used to adjust position.

The Prediction Algorithm

- Two main algorithms commonly used in location based prediction are:
  - The first algorithm predicts on the basis of the unit's last location and its velocity at that time
  - The second predicts on the basis of the unit's last known position, its velocity and acceleration at that time.
- More sophisticated algorithms make use of the roll, pitch, direction and may also do predictions on different parts of the unit independently.

Prediction Cont'd

- Assuming x(t) is the position at time t and the last update for a unit's position was received at time t_0. With no opponent prediction, a client will assume the location of a unit at time t_1 to be the same as the location of the unit at time t_0 .

$$x(t_1) = x(t_0)$$

- Now assuming a constant velocity (v), a simple prediction algorithm would predict the location of the unit to be:

$$x(t_1) = x(t_0) + v * (t_1 - t_0)$$

Prediction Cont'd

- Adding information about a constant acceleration (a), the location of the unit would be predicted to be:

$$x(t_1) = x(t_0) + v * (t_1 - t_0) + (1/2)a * (t_1 - t_0)^2$$

- In general, units with high inertia are easier to predict (like a rolling stone) than units with lower inertia (a person walking).

Time Manipulation

- The game states rendered at the clients are different because a server response takes some time to reach the client machine depending on the location of the client from the server.
- Thus, when the same response reaches players at different positions in time, there is an unfairness factor introduced in the game.
- Two efficient techniques for handling this manipulation are *Time Delay* and *Time Warp*.

Time Delay

- In this technique there is a delay in processing and sending of user commands to equalize latency.
- Instead of processing and responding to the client request right away, the server purposely introduces a delay, thus allowing a remote client to respond to the game state.
- This technique ensures that all clients have the same effective latency in the game world at the server – consistent game view

Time Delay Cont'd

- The world state updates sent by the server can also be delayed in being ***sent out***, sending the response to a client that is far away before sending the response to a client that is closer.
- The clients themselves can introduce ***buffering*** to equalize latency, with the client that is closer to the server delaying the processing of the server response while the one which is further away can process the response immediately.
- But while buffering works towards equalizing the latency among various clients, it makes the game play less responsive.

Time Warp

- Another mechanism for time manipulation is to have a server rollback or *Time Warp* the events in a game to the point when a client command was input.
- *For example:* the player shoots at an opponent at time t_0, but by the time the message arrives at the server at time t_2 , the opponent moved at time t_1. The server uses the time warp technique and rolls back the events it had processed since the client provided the input.
- In this case, the server might determine that this older event has a bearing on subsequent events, changing their effect to make the global world state consistent. In other words, the server may determine that the player hit and killed the opponent, meaning that the opponent's movement at time t_1 was invalid.

Time Warp

- General Algorithm for the Server:
  - Receive packet from client
  - Extract information (user input)
  - Elapsed time = current time – latency to client
  - Rollback all events in reverse order to current time – elapsed time
  - Execute user command
  - Repeat all events in order, updating any clients that are affected
  - Repeat.

Time Warp Cont'd

- For Time Warp it is critical to have an accurate measurement of the latency between a client and a server so that the game time can be rolled back the exact amount of time.
- The popular first person shooter game, Half-Life 2 (HL2), makes use of time warp. For testing purposes, the HL2 server allows additional lag to be added to the clients. The server administrator can observe the actual location of a unit and the location for the unit with time warp by having a separate client on the same machine as the server (a listen server).

Time Warp Cont'd

- Figure shows a screenshot of an HL2 listen server.
- Round-trip latency to the client - 200 ms, meaning the user's commands are executed 100 ms before the screenshot.
- Grey boxes show the target position on the client where it was 100 ms ago.
- Since then, the target moved to the left while the user's command was travelling over the network to the server.
- When the user command arrives at the server, the server rolls back time (time warp) to put the target in the position it was at the time the user shot, indicated by the black boxes. The server determines there was a hit (the client sees blood from the wounds).

Time Warp Shortcomings

- Can cause inconsistencies
  - Example: Suppose a player places the cross-hairs of a gun on an opponent and fires. The server, using time warp, will ultimately determine this is a “hit”. However, in the meantime, because of client–server latency, the opponent may have moved, perhaps even around a corner and out of sight. When the server warps time back to when the shot was fired and determines the opponent was shot, it will seem to the opponent that the bullets actually went around the corner.

Data Compression

- It is evident that reducing the size of the message sent between a server and a client can also reduce latency.
- A smaller packet has shorter transmission time than a larger packet.
- Types of compression:
  - Lossless Compression
  - Opponent Prediction
  - Delta Compression
  - Interest Management
  - Peer to Peer
  - Update Aggregation

Visual Tricks

- These are some additional techniques which do not involve networking, but are used to cover up network latency on the client side.
- A start-up animation could be used to hide latency from the client to the server.
  - Example: When a boat is ready to move, the game may require it to raise sails before it starts to actually move. Such animations delays can take a couple of hundred milliseconds, allowing the message to go to the server and back before the unit actually moves.
  - Thus, if a server indicates the move is not allowed due to any reason, there is no discrepancy to fix, and the player feels that the game is very responsive.
- Local confirmation of an action having taken place can be used immediately even if the remote effect is not confirmed by the server.
  - Example: If a player puts his finger on the gun, the game client can play a shooting sound effect and show a smoke puff, even though the ***effect*** of the shot is not determined for some time.

Latency Compensation and Cheating

- The anonymity factor in online games means there are many opportunities for cheating.
- Some of the latency compensation techniques while reducing the effect of latency introduce more opportunities for cheating.

Latency Compensation and Cheating (Cont.)

- Cheating Opportunities Examples:
  - With time warp, a client could interfere with measurements about round-trip time, making the server believe the client is further away than it actually is. This would allow the client to respond to events that, in essence, happened in the past and hence giving unfair advantage.
  - With time delay, a client with slow reflexes could claim a higher latency than it actually has, causing a large time delay at the server, thus neutralizing the better reflexes that opponents may have.
  - Interest management, while reducing network bitrates, can also be abused by cheaters. Clients can claim interest in game state that they could otherwise not see, using this information to gain a tactical advantage.
- Server Control:
  - This justifies the need of an authoritative server which could confirm or refute inappropriate client requests and actions and also keep track of the exact time delay between the server and each client.