Distributed Computing Systems

Network Games

References / Reading

- [BT01] P. Bettner and M. Terrano. 1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond, *Gamasutra*, March 22, 2001
- [SKH02] J. Smed, T. Kaukoranta and H. Hakonen. Aspects of Networking in Multiplayer Computer Games, *The Electronic Library*, Volume 20, Number 2, Pages 87-97, 2002
- [CFGS14] Mark Claypool, David Finkel, Alexander Grant and Michael Solano. On the Performance of OnLive Thin Client Games, *Springer Multimedia Systems Journal (MMSJ) - Special Issue on Network Systems Support for Games*, pages 1-14, February 2014.

Outline

- Synchronization in AoE (next)
- Aspects of Networking
- Cloud Games

Age of Empires – Real Time Strategy

- Build
- Combat
- Explore

Microsoft Studios, 1997

AoE: Multiplayer Design Goals

- Wanted: army on army, large supporting structure, … ("1500 archers on a …")
- Support for 8 players
- Smooth simulation over modem, Internet, LAN
- Target platform: 16 MB P-90, 28.8 modem
- 15 frames per second (one frame every 67 ms)
- Use (existing) Genie engine
  - 2d, sprites in 256 colors
  - Reasonably stable

AoE in Early Stages

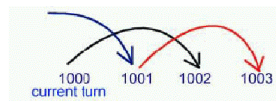
- Engine performance breakdown:
  - 30% graphic rendering
  - 30% AI
  - 30% simulation
- Time to complete each simulation step varied:
  - Render time changes with number of units
  - When scrolling
  - AI computation time varied with units or time
  - As much as 200 ms (larger than a frame time!)
- Bandwidth a critical resource:
  - Passing (x,y) coordinates, status, action, facing damage … limit of 250 moving units at most

Simultaneous Simulations

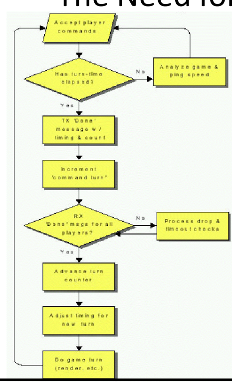
- Each PC ran exact same simulation
  - Synchronized game time
  - Synchronized random number generators
- Still
  - Internet latency from 20 to 1000 milliseconds
  - Variable time to process each step
- Needed more responsive approach

Communication Turns

- Separate communications turns from frame rendering
- Schedule commands for later time
  - Allows for some variance in network and turn processing
- Turns typically 200 ms in length
  - Send all commands entered that turn, but schedule them for 2 turns later
  - Process any scheduled turns



The Need for Speed Control

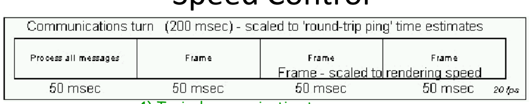


- Since all machines in "lock step", can only run as fast as slowest machine
  - Process communications, render turn, send out new commands
- "Lag" if
  - One machine slows down and others wait
  - Delayed or lost Internet data

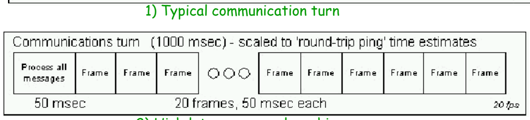
Speed Control

- Each client calculates frame rate
  - Since varies with game state, use moving average
  - Send with "Turn Done" message
  - Use to achieve "minimum" frame rate
- Each client measures round-trip "ping" time
  - Since varies with Internet traffic, use largest for all players
- After getting "Turn Done" messages
  - Adjust target frame rate (based on local PC render rate)
  - Adjust communication turn (based on ping-times + remote PC render rates)
  - Weighted, so only "laggy" during worst spikes
- (Examples next)

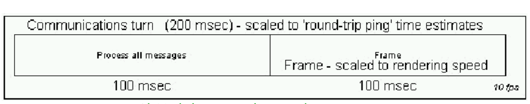
Speed Control



1) Typical communication turn



2) High latency, normal machine



3) High latency, slow machine

Transport Protocol - UDP

- Unreliable, so each client handles command ordering, drop detection and re-sending
  - "When in doubt, assume it dropped"
- Messages arriving from past turns are discarded
- If out of order message received, request a resend of supposedly "missing" messages
  - Note, if really out of order, will get duplicate so must account for
- If ack is "late", then assume lost so resend

Side Benefit – Cheat Prevention

- Simultaneous simulations means games are identical
- If there is a discrepancy, game stopped
- Prevents cheaters from using hacked client
- But there still could be cheating via information exposure

Side Problems – Out of Synch

"In every project, there is one stubborn bug that goes all the way to the wire…"
– Microsoft product manager

- Subtle, since small errors multiply
  – Example – a deer slightly out of alignment, causes villager to "miss" so no meat, causing different food amounts
- Checksums (objects, pathing, targeting …), but always *something*
  – Wade through 50 MB of message traces

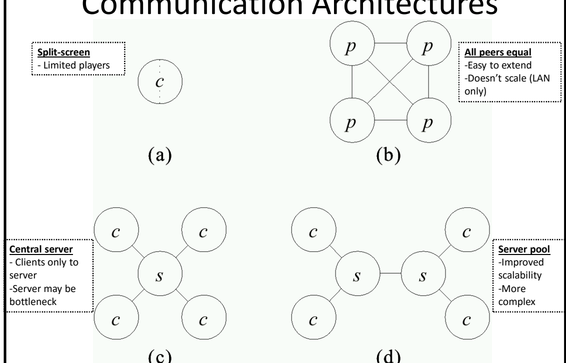
Outline

- Synchronization in AoE (done)
- Aspects of Networking (next)
- Cloud Games

Network Latency

- Delay when message sent until received
  – Variation in delay (delay jitter) also matters
- Cannot be totally eliminated
  – e.g., speed of light propagation yields 25-30 ms across Atlantic
  – And with routing and queuing, usually 80+ ms
- Application tolerances:
  – File download – minutes
  – Web page download – up to 10 seconds
  – Interactive audio – 100s of ms
- MCG latencies tolerance? → Depends upon game!
  – First-Person Shooters – about 100 ms
  – Third-Person Adventure – up to 500 ms
  – Real-Time Strategy – up to 1 second
  – And depends upon action *within* game! (topic for another paper)

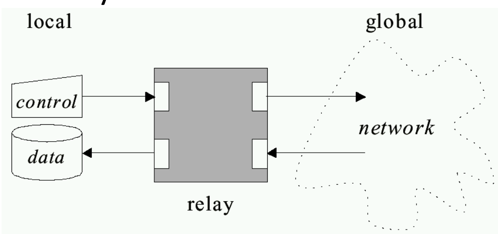
Communication Architectures



Data and Control Architectures

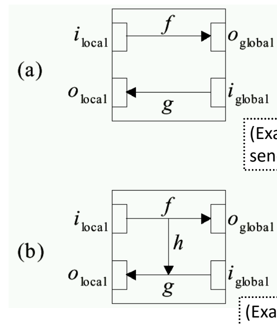
- Want *consistency*
  – Same state on each node
  – Needs tightly coupled, low latency, small nodes
- Want *responsiveness*
  – More computation locally to reduce network
  – Loosely coupled (asynchronous)
- In general, cannot do both → *Tradeoffs*

“Relay” Architecture Abstraction



- Want control to propagate quickly so can update data (*responsiveness*)
- Want to reflect same data on all nodes (*consistency*)

Relay Architecture Choices



(a)
$$o_{\text{global}} = f(i_{\text{local}})$$
$$o_{\text{local}} = g(i_{\text{global}})$$

(Example: Dumb terminal, send and wait for response)

(b)
$$o_{\text{global}} = f(i_{\text{local}})$$
$$o_{\text{local}} = g(i_{\text{global}}) \times h(i_{\text{local}})$$

(Example: Smart terminal, send and echo)

MCG Architectures

- *Centralized*
  - Use only two-way relay (no short-circuit)
  - One node holds data so view is consistent at all times
  - Lacks responsiveness
- *Distributed* and *Replicated*
  - Allow short-circuit relay
  - Replicated has copies, used when predictable (e.g., behavior of non-player characters)
  - Distributed has local node only, used when unpredictable (e.g., behavior of players)

Compensatory Techniques

- Architectures alone not enough
- Design to compensate for residual
- Techniques:
  - Message aggregation
  - Interest management
  - Dead reckoning

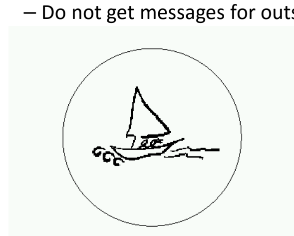
    (next)

Message Aggregation

- Combine multiple messages in one packet to reduce network overhead
- Examples:
  - Multiple user commands to server (move *and* shoot)
  - Multiple users command to clients (player A’s *and* player B’s actions combined to player C)

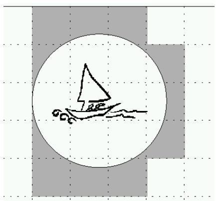
Interest Management – Auras (1 of 2)

- Nodes express area of interest to them
  - Do not get messages for outside areas

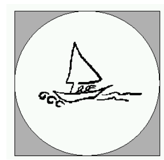


- Only circle sent even if world is larger
- But implementation complex (squares easier)

Interest Management- Auras (2 of 2)



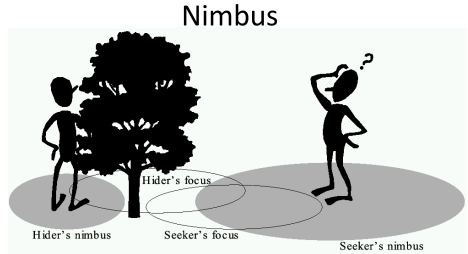
- Divide into cells (or hexes)
- Easier, but less discriminating



- Compute bounding box
- Relatively easy, precise

- Always symmetric – both receive
  - But can sub-divide – *focus* and *nimbus*

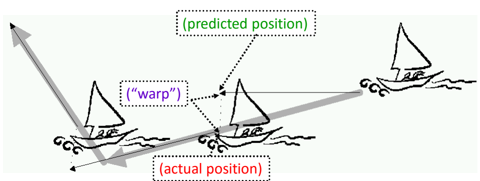
Interest Management- Focus and Nimbus



- *Nimbus* must intersect with *focus* to receive
- Example above: hider has smaller nimbus, so seeker cannot see, while hider can see seeker since seeker's nimbus intersects hider's focus

Dead Reckoning

- Based on ocean navigation techniques ("dead" == "deduced (ded.)")
- Predict position based on last known position plus direction
  - Only send updates when deviates past threshold



- When prediction differs and adjust, get "warping" or "rubber-banding" effect
  - Some techniques move to place over short time

Serial and Parallel Execution

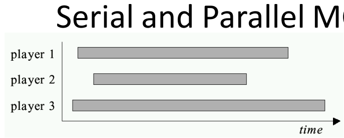
- Given time T(1), speedup with *n* nodes $S(n) = \frac{T(1)}{T(n)} \leq \frac{1}{n}$
- Part of T(1) must happen serially and part can be done in parallel
  $T_s + T_p = T(1)$ and $\alpha = T_s/(T_s + T_p)$
- If serialized optimally:

(Amdahls' law)

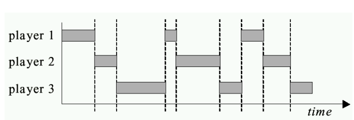
$$S(n) = \frac{T_s + T_p}{T_s + T_p/n} = \frac{1}{\alpha + (1-\alpha)/n} \leq \frac{1}{\alpha}.$$

- If $T_s = 0$, everything parallelizable but then no communication
  (ex: players at own console with no interaction)
- If $T_p = 0$, then turn based
- Between are MCGs which have some of both

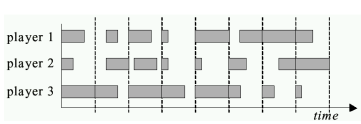
Serial and Parallel MCGs



Separate games



Turn-based games



Interactive games

Communication Capacity

- Scalability limited by communication requirements of chosen architecture

Deployment architecture	Capacity requirement
Single node	0
Peer-to-peer (Multicasting)	$\sim n \ldots n^2$
Client/server	$\sim n$
Peer-to-peer server-network	$\sim \frac{n}{m} + m \ldots \frac{n}{m} + m^2$
Hierarchical server-network	$\sim n$

- Can consider pool of *m* servers with *n* clients divided evenly amongst them
- Servers in hierarchy have root as bottleneck
- In order not to increase with *n*, must have clients not aware of other clients (interest management) and do message aggregation

Cheating

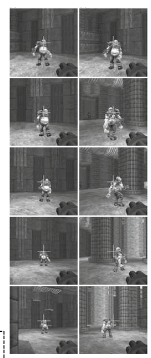
- Unique to games
  - Other multi-person applications don't have
  - In DIS, military not public and considered trustworthy
- Cheaters want:
  - *Vandalism* – create havoc (relatively few).
    - Mostly, game design to prevent (e.g., no friendly fire)
  - *Dominance* – gain advantage (more)
    - Next slides

Packet and Traffic Tampering

- *Packet interception* – prevent some packets from reaching cheater
  - e.g., suppress damage packets, so cheater is invulnerable
- *Packet replay* – repeat event over for added advantage
  - e.g., multiple bullets or rockets if otherwise limited
- Solutions:
  - MD5 Checksum or Encrypt packets
  - Authoritative host keeps within bounds

Packet Tampering

- *Reflex augmentation* - enhance cheater's reactions
  - e.g., aiming proxy monitors opponents movement packets, when cheater fires, improve aim
- Tough to detect
  - E.g., PunkBuster – scan for "known" hacks
  - False positives?

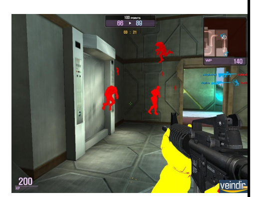


aimbot human

S. Yeung and J. Lui. "Dynamic Bayesian approach for detecting cheats in multi-player online games", Springer Multimedia Systems, Vol. 14, No. 4 Sep. 2008.

Information Exposure

- Allows cheater to gain access to replicated, hidden game data (e.g. status of other players)
  - Passive, since does not alter traffic
  - e.g., ignore "fog of war" in RTS, or "wall hack" to see through walls in FPS
- Cannot be defeated by network alone
- Instead:
  - Sensitive data should be encoded
  - Kept in hard-to-detect memory location
  - Centralized server may detect cheating (e.g., attack enemy could not have seen)



Outline

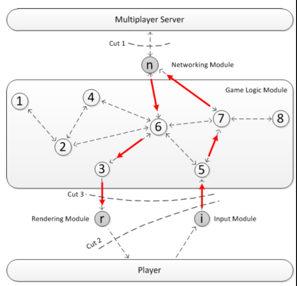
- Synchronization in AoE (done)
- Aspects of Networking (done)
- Cloud Games (next)

Why Games as a Service?

- Potential scalability
  - Overcome processing and storage limitations
- Cross-platform support
  - Can run games built for different platforms (e.g., Xbox and Playstation) on one device
- Piracy prevention
  - Since game code is stored in cloud, cannot be copied
- Click-to-play
  - Game can be run without installation

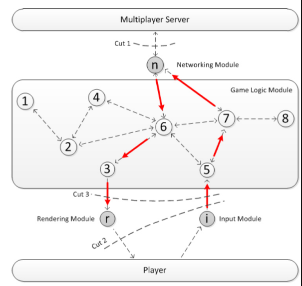
Cloud Game Modules (1 of 2)

- *Input* – receives control messages from players
- *Game logic* – manages game content
- *Networking* – exchanges data with server
- *Rendering* – renders game frames
- How do put in cloud?

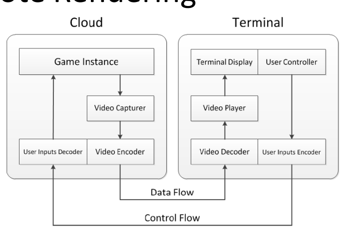


Cloud Game Modules (2 of 2)

- Cuts
  1. All game logic on player, cloud only relay information (traditional network game)
  2. Player only gets input and displays frames (remote rendering)
  3. Player gets input and renders frames (local rendering)



Remote Rendering

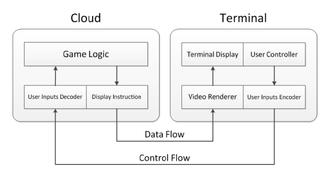


- Cloud runs full, traditional game
- Captures video ("scrape" screen) and encode
- Client only needs capability to decode and play
  - Relatively minor requirements
- Bitrate requirements can be an issue

- e.g.,
  - Onlive (commercial)
  - Gaming Anywhere (research)
  - Cloud Saucer Shoot (teaching)

Local Rendering

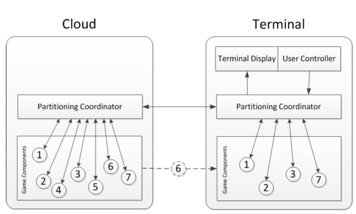
- Instead of video frames, send display instructions
  - Potentially great bitrate savings
- Challenge for instruction set: able to represent all images for all games



e.g., Browser-based games (via HTML5 and/or Javascript), [De Winter et al., NOSSDAV '06]

Potential Distribution of Computing

- Partitioning coordinator if/when to migrate functionality (e.g., reduce cloud load and/or when terminal has greater capabilities)
  - Remote and Local rendering cases (above) are really just special cases
- Challenge: how to do so in general, how to synchronize if both cloud + terminal have module (e.g., "6")

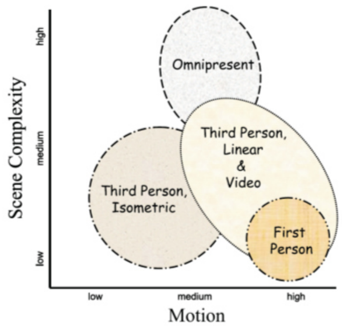


e.g., [Cai et al., CloudCom 2013]

Application Streams vs. Game Streams

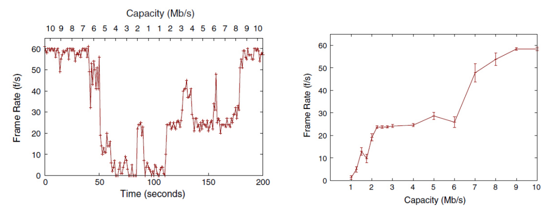
- Traditional thin client applications (e.g., x-term, remote login shell):
  - Relatively casual interaction
    - e.g., typing or mouse clicking
  - Infrequent display updates
    - e.g., character updates or scrolling text
- Computer games:
  - Intense interaction
    - e.g., avatar movement and shooting
  - Frequently changing displays
    - e.g., 360 degree panning

Motion and Scene Complexity - Summary



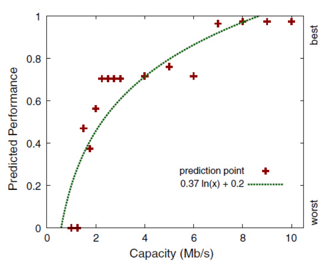
Downstream Bitrate

Capacity restriction



Capacity affects frame rate
(OnLive recommends 5 Mb/s, but accepts 2 Mb/s)

Predicted Player Performance



(Model based on FPS data with restricted frame rates)
Capacity affects performance

Network Turbulence Summary

Application	Bitrate (kb/s)	Pkt size (bytes)	Inter-Pkt (ms)
Traditional game	67	75	45
Virtual environment	775	1,027	9
Live video	2,222	1,314	0.1
Thin Game	6,247	1,203	0.7
Pre-recorded Video	43,914	1,514	0.1

Cloud-Game Summary

- Games as service new model for cloud computing
  - Choices on distribution of rendering and computation
- Cloud games are like video, but different
  - Wider range of motion and scene complexity
- OnLive
  - Like video conference down, traditional games up
  - Bitrate responds to capacity, but not loss or latency
    - Not TCP-Friendly
  - Best for players above 5 Mb/s, with 2 Mb/s minimum
    - Lower capacities affect player performance