1500 Archers on a 28.8: Network Programming in Age of Empires and Beyond

J. Smed, T. Kaukoranta and H. Hakonen

Gamasutra
(First appeared at the *Game Developer's Conference, 2001)*
March 22, 2001

Age of Empires – Promo

http://www.microsoft.com/games/empires/multimedia.htm

Age of Empires – Real Time Strategy

- Build
- Combat
- Explore

Introduction

- This page explains Age of Empires (AoE) 1 and 2 multiplayer implementation.
- Explains:
  - Design Architecture
  - Implementation
  - Lessons learned
- Also for the "future" RTS by Ensemble
  - (Age of Mythology, AoM)

Outline

- Introduction (done)
- Implementation (next)
- Lessons Learned
- Improvements for AoE 2
- RTS3
- Summary

AoE: Multiplayer Design Goals

- Support for 8 players
- Smooth simulation over modem, Internet, LAN
- Target platform: 16 MB P-90, 28.8 modem
  - (AoM is PX-450, 56.6 modem)
  - At 15 frames per second (one frame every 67 ms)
- Use (existing) Genie engine
  - next

AoE in Early Stages (1 of 2)

- Game engine was running
  - 2d, single-threaded game loop
  - Sprites in 256 colors
  - Reasonably stable
- Large map, thousands of objects, trees could be chopped, animals ran around …
- Breakdown:
  - 30% graphic rendering
  - 30% AI
  - 30% simulation
- "Compelling" single-player game already
  - (ie- "Don't ruin it!")

AoE in Early Stages (2 of 2)

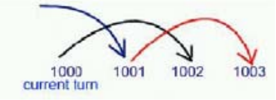
- Wanted: army on army, large supporting structure, … ("1500 archers on a …")
- Time to complete each simulation step varied:
  - Render time changes with number of units
  - When scrolling
  - AI computation time varied with units or time
  - As much as 200 ms (larger than a frame time!)
- Bandwidth a critical resource:
  - Passing x,y coordinates, status, action, facing damage … limit of 250 moving units at most
  - (MLC: 1 bytes each → 6 actions x 250 units x 15 updates per second ~ 160 Kbps)

Simultaneous Simulations

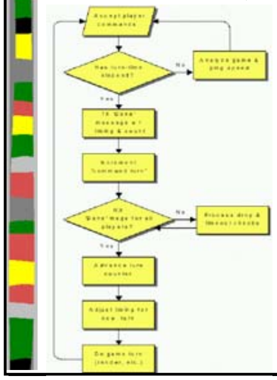
- Each PC ran exact same simulation
  - Synchronized game time
  - Synchronized random number generators
- Still
  - Internet latency from 20 to 1000 milliseconds
  - Variable time to process each step
- Needed a more responsive approach

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

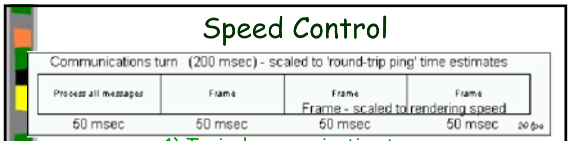

Accept player commands
Has turn-time elapsed?
No
Analyze game & ping speed
Yes
TX 'Done' message w/ timing & count
Increment "command turn"
All 'Done' msgs for all players?
No
Process drop & timeout checks
Yes
Advance turn counter
Adjust timing for new turn
Do game turn (render, etc.)

- 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

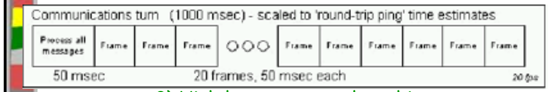
Speed Control

- 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 (MLC: assume moving average)
- fps + rtt → 2-bytes total overhead
- 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)

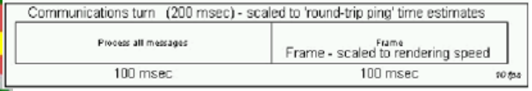
Speed Control


Communications turn (200 msec) - scaled to 'round-trip ping' time estimates
Process all messages
Frame
Frame
Frame
Frame - scaled to rendering speed
50 msec
50 msec
50 msec
50 msec
20 fps

1) Typical communication turn



2) High latency, normal machine


Communications turn (200 msec) - scaled to 'round-trip ping' time estimates
Process all messages
Frame
Frame - scaled to rendering speed
100 msec
100 msec
10 fps

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
  - (MLC: different game states when commands are lost or are too late?)

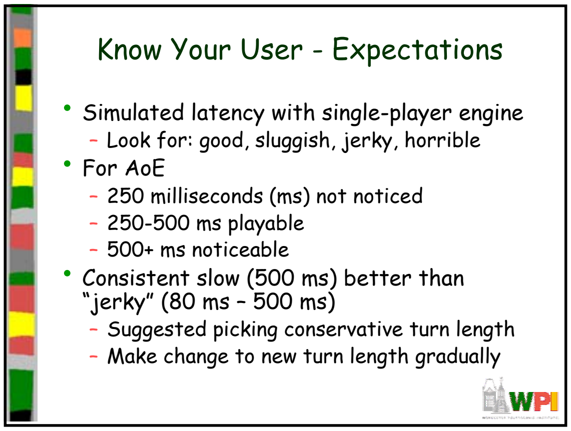
Outline

- Introduction (done)
- Implementation (done)
- Lessons Learned (next)
- Improvements for AoE 2
- RTS3
- Summary

Lesson: Know Your User

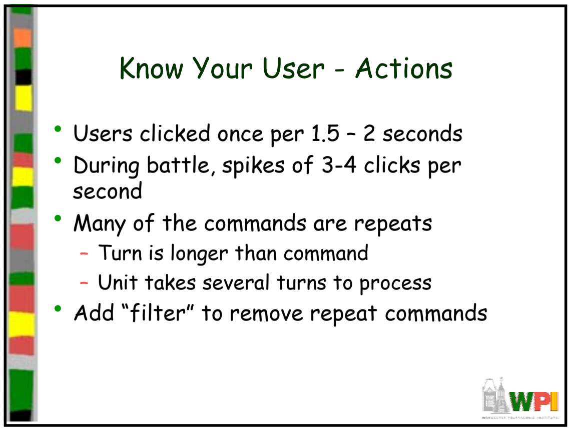
- Each genre is different
  - RTS
  - FPS
  - Sports
  - MMORPG
  - …

1) Know latency expectations
2) Prototype multiplayer aspects early
3) Watch for behavior that hurts multiplayer performance



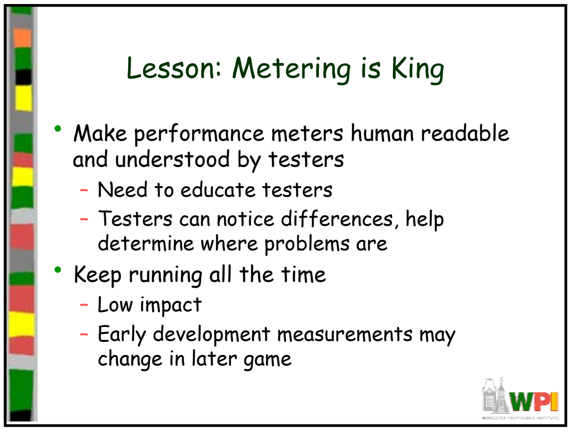
Know Your User - Expectations

- Simulated latency with single-player engine
  - Look for: good, sluggish, jerky, horrible
- For AoE
  - 250 milliseconds (ms) not noticed
  - 250-500 ms playable
  - 500+ ms noticeable
- Consistent slow (500 ms) better than "jerky" (80 ms – 500 ms)
  - Suggested picking conservative turn length
  - Make change to new turn length gradually



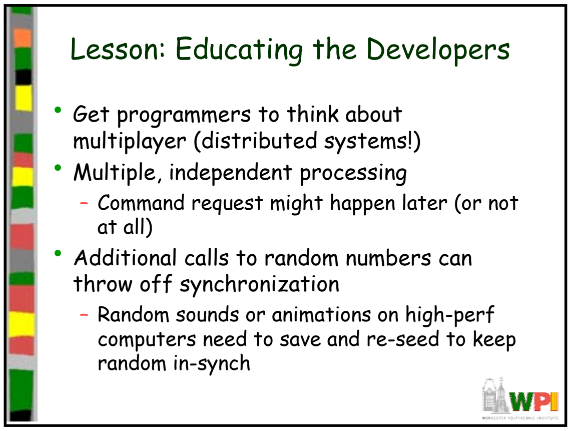
Know Your User - Actions

- Users clicked once per 1.5 – 2 seconds
- During battle, spikes of 3-4 clicks per second
- Many of the commands are repeats
  - Turn is longer than command
  - Unit takes several turns to process
- Add "filter" to remove repeat commands



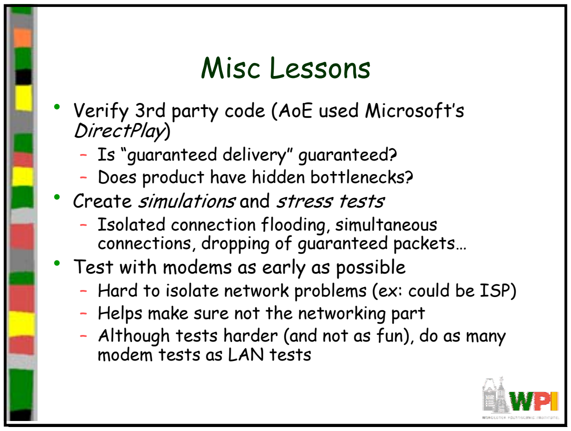
Lesson: Metering is King

- Make performance meters human readable and understood by testers
  - Need to educate testers
  - Testers can notice differences, help determine where problems are
- Keep running all the time
  - Low impact
  - Early development measurements may change in later game



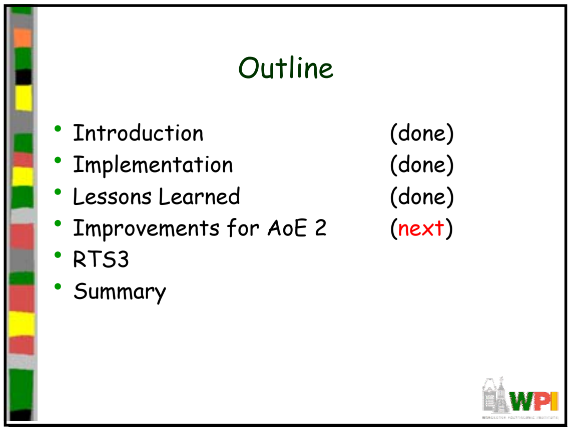
Lesson: Educating the Developers

- Get programmers to think about multiplayer (distributed systems!)
- Multiple, independent processing
  - Command request might happen later (or not at all)
- Additional calls to random numbers can throw off synchronization
  - Random sounds or animations on high-perf computers need to save and re-seed to keep random in-synch



Misc Lessons

- Verify 3rd party code (AoE used Microsoft's *DirectPlay*)
  - Is "guaranteed delivery" guaranteed?
  - Does product have hidden bottlenecks?
- Create *simulations* and *stress tests*
  - Isolated connection flooding, simultaneous connections, dropping of guaranteed packets…
- Test with modems as early as possible
  - Hard to isolate network problems (ex: could be ISP)
  - Helps make sure not the networking part
  - Although tests harder (and not as fun), do as many modem tests as LAN tests



Outline

- Introduction (done)
- Implementation (done)
- Lessons Learned (done)
- Improvements for AoE 2 (next)
- RTS3
- Summary

Improvements for Age of Empires 2 – the Age of Kings

- New multiplayer features
  - Recorded games
  - File transfer (custom maps)
  - Statistics tracking (on "The Zone")
- Recorded games helps in debugging
  - Can replay exactly to spot problems

Outline

- Introduction (done)
- Implementation (done)
- Lessons Learned (done)
- Improvements for AoE 2 (done)
- RTS3 (next)
  - Overview
  - New features and tools
- Summary

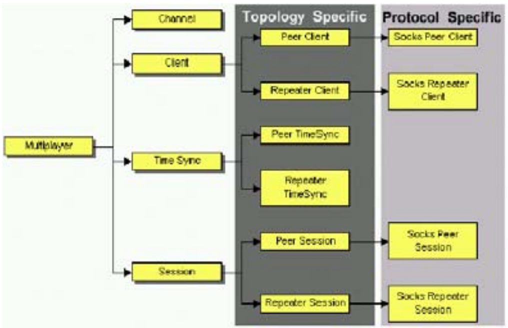
RTS3 – Beyond AoE

- RTS3 is "codename" for next generation Ensemble RTS (probably Age of Mythology)
- Add 3-d capability (used BANG!)
- Multiplayer requirements
  - Again, complex maps, thousands of units, Internet play
  - More than 8 players (AoM allows 12)
  - Still modem, but 56.6k
  - May be *firewalls* and *NAT boxes* so peer-to-peer harder

RTS3

- Forget DirectPlay → created own library
  - Use in subsequent games
  - Integrated with BANG!
- Fully 3-d world meant frame rate maybe an issue
  - Overall smoothness from frame rate impacted
  - Also more variation
- Realized play-testing was iterative, so wanted multiplayer running earlier

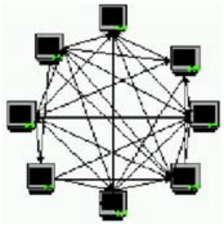
An OO Approach

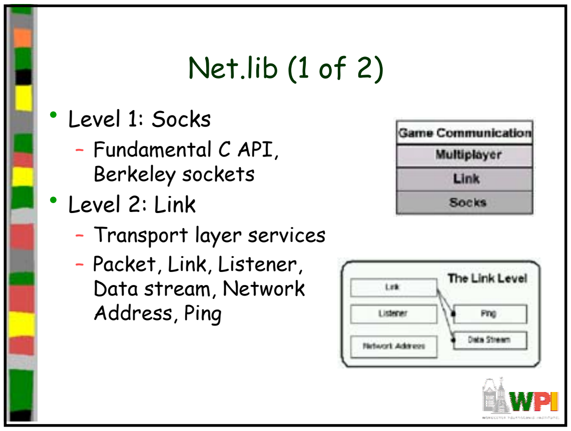


(Make protocol specific parts as small as possible)

Peer-to-Peer Topology

- Strengths
  - Reduces latency
  - No central point of failure (can continue game if one client leaves)
- Weaknesses
  - More active connections (n)
  - Impossible to support some NATs

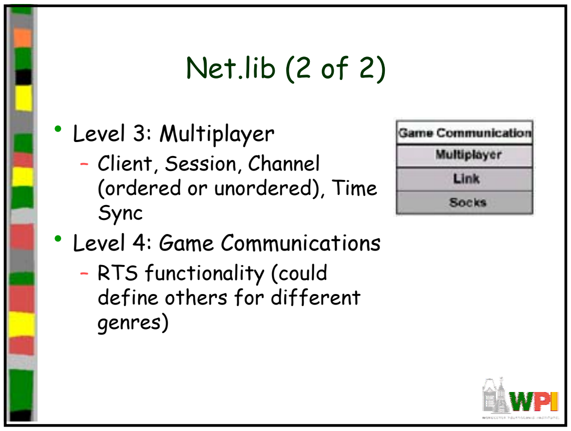




Net.lib (1 of 2)

- Level 1: Socks
  - Fundamental C API, Berkeley sockets
- Level 2: Link
  - Transport layer services
  - Packet, Link, Listener, Data stream, Network Address, Ping

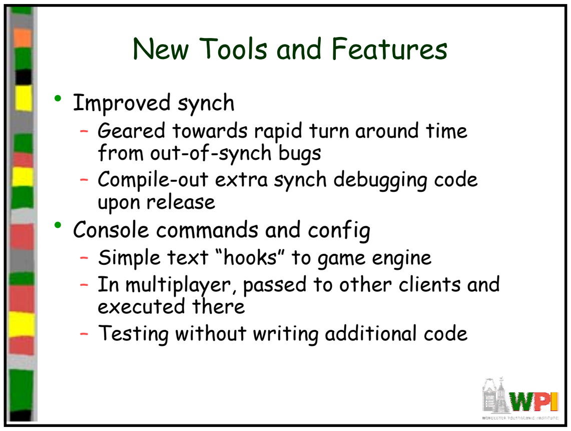
Game Communication
Multiplayer
Link
Socks
The Link Level
Link
Listener
Network Address
Ping
Data Stream



Net.lib (2 of 2)

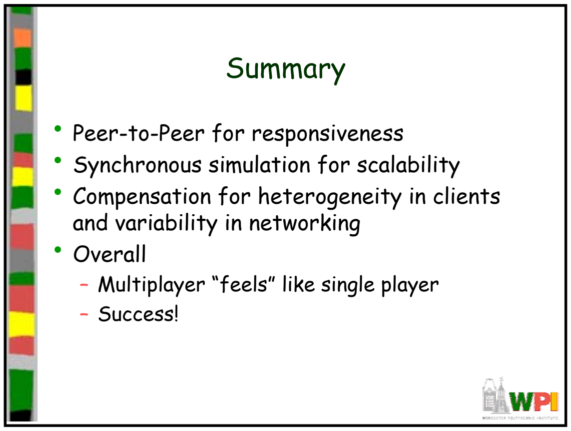
- Level 3: Multiplayer
  - Client, Session, Channel (ordered or unordered), Time Sync
- Level 4: Game Communications
  - RTS functionality (could define others for different genres)

Game Communication
Multiplayer
Link
Socks



New Tools and Features

- Improved synch
  - Geared towards rapid turn around time from out-of-synch bugs
  - Compile-out extra synch debugging code upon release
- Console commands and config
  - Simple text “hooks” to game engine
  - In multiplayer, passed to other clients and executed there
  - Testing without writing additional code



Summary

- Peer-to-Peer for responsiveness
- Synchronous simulation for scalability
- Compensation for heterogeneity in clients and variability in networking
- Overall
  - Multiplayer “feels” like single player
  - Success!