Aspects of Networking in Multiplayer Computer Games

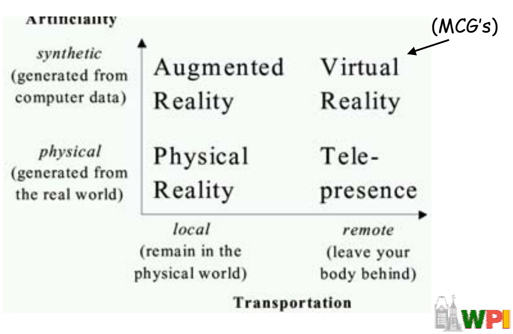
J. Smed, T. Kaukoranta and H. Hakonen

The Electronic Library
Volume 20, Number 2, Pages 87-97
2002

Introduction

- Internet + wireless making multiplayer computer games (MCGs) more popular
- Commercial computer games increasingly having mutiplayer option. With servers:
  - Electronic Arts – *Ultima Online*
  - Blizzard – *Battle.net*
  - Microsoft's – *MSN Gaming Zone*
- Consoles, too (*PS2*, *Xbox*)
- Wireless devices, too (*Nokia NGage*)

Shared Space Technologies



Other VR Research Efforts

- *Distributed Interactive Simulations* (DIS)
  - Protocol (IEEE), architectures …
  - Ex: flight simulation
  - Large scale, spread out, many users
- *Distributed Virtual Environments* (DVEs)
  - Immersive, technology oriented
  - Ex: "Caves"
  - Local, few users
- *Computer Supported Cooperative Work* (CSCW)
  - Focus on collaboration
  - Ex: 3D editors
- And MCGs are similar, yet not discussed in scientific literature → Hence, this paper seeks to rectify

Outline

- Introduction (done)
- Networking Resources (next)
- Distribution Concepts
- Scalability
- Security and Cheating
- Conclusions

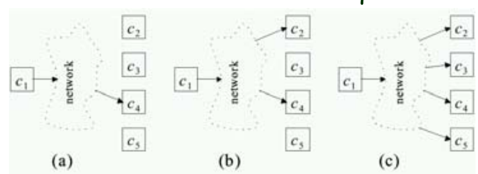
Network Resources

- Distributed simulations face three resource limitations
  - *Network bandwidth*
  - *Network latency*
  - *Host processing power* (to handle network)
- Physical restrictions that the system cannot overcome
  - Must be considered in the design of the application
- (More on each, next)

Bandwidth (Bitrate)

- Data sent/received per time
- LAN – 10 Mbps to 10 Gbps
  - Limited size and scope
- WANs – tens of kbps from modems, to 1.5 Mbps (T1, broadband), to 55 Mbps (T3)
  - Potentially enormous, Global in scope
- Number of users, size and frequency of messages determines bitrate use
- As does transmission technique (next slide)

Transmission Techniques



- (a) Unicast, one send and one get
  - Wastes bandwidth when path shared
- (c) Broadcast, one send and all get
  - Perhaps ok for LAN
  - Wastes bandwidth when most don't need
- (b) Multicast, one send and only subscribed get
  - Current Internet does not support
  - Multicast *overlay* networks

Network Latency

- Delay when message sent until received
  - Variation (jitter) also matters
- Cannot be totally eliminated
  - Speed of light propagation yields 25-30 ms across Atlantic
  - 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 – 100s of ms
  - Real-Time Strategy – up to 1 second [SGB+03]
  - Other games

Computational Power

- Processing to send/receive packets
- Most devices powerful enough for raw sending
  - Can saturate LAN
- Rather, *application* must process state in each packet
- Especially critical on resource-constrained devices
  - I.e.- hand-held console, cell phone, PDA,

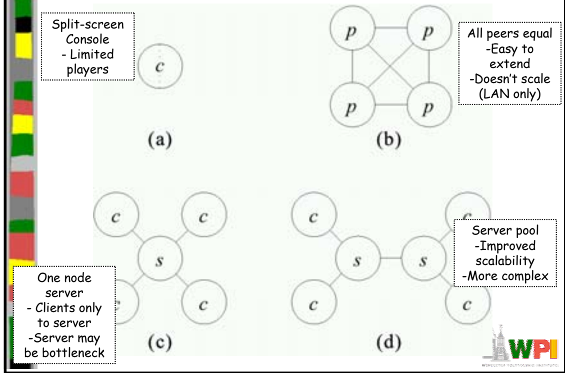
Outline

- Introduction (done)
- Networking Resources (done)
- Distribution Concepts (next)
- Scalability
- Security and Cheating
- Conclusions

Distribution Concepts

- Cannot do much about above resource limitations
- Should tackle problems at higher level
- Choose architectures for
  - *Communication*
  - *Data*
  - *Control*
- Plus, *compensatory techniques* to relax requirements

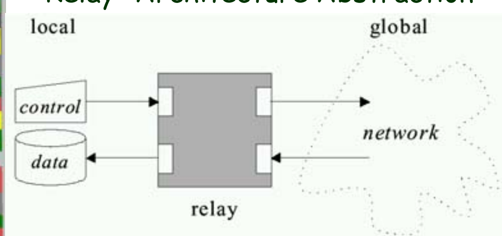
Communication Architectures



Data and Control Architectures

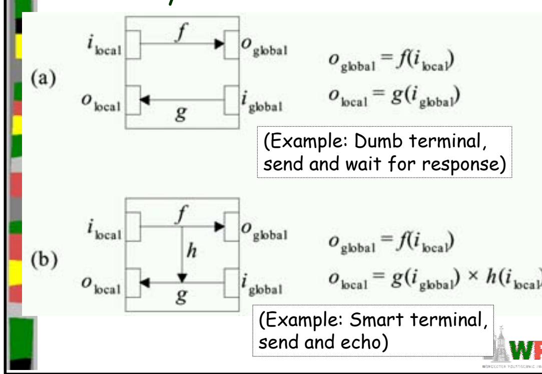
- Want *consistency*
  - Same state on each node
  - Needs tightly coupled, low latency, small nodes
- Want *responsiveness*
  - More computation locally to reduce network
  - Loosely coupled
- In general, cannot do both. Tradeoffs.

"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



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 (ie- non-player characters)
  - Distributed has local node only, used when unpredictable (ie- players)

Compensatory Techniques

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

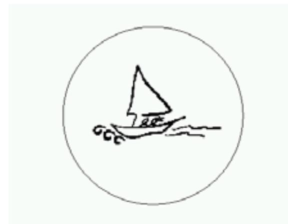
(next)

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 to player C)

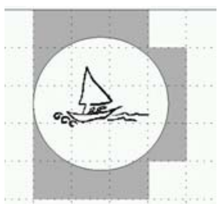
Interest Management – Auras (1)

- 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

Interest Management- Auras (2)

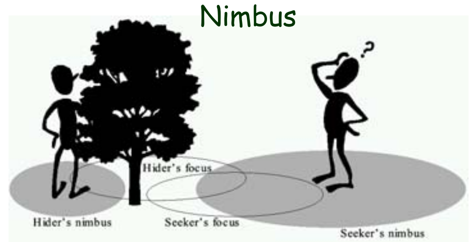


- Divide into cells (or hexes).
- Easier, but less discriminating

- Compute bounding box
- Relatively easy, precise

- Always symmetric – both receive
  - But can sub-divide – *Focus* and *Nimbus*

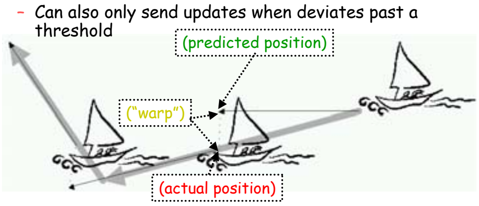
Interest Management- Focus and Nimbus



-*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
- Predict position based on last known position plus direction
  - Can also only send updates when deviates past a threshold



- When prediction differs, get "warping" or "rubber-banding" effect

Outline

- Introduction (done)
- Networking Resources (done)
- Distribution Concepts (done)
- Scalability (next)
- Security and Cheating
- Conclusions

Scalability

- Ability to adapt to resource changes
- Example:
  - Expand to varying number of players
  - Allocate non-player computation among nodes
- Need hardware parallelism that enables software concurrency

Serial and Parallel Execution

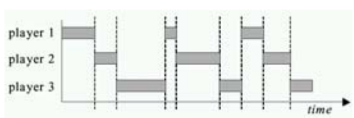
- Given time T(1), speedup with *n* nodes $S(n) = \frac{T(1)}{T(n)} \leq \frac{1}{n}$
- Part of T(1) is serializable, part is parallel
  - $T_s + T_p = T(1)$ and $\alpha = T_s/(T_s + T_p)$
- If serialized optimally: *(Amdahls' law)*

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

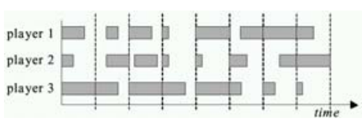
- If $T_s = 0$, everything parallelizable but then no communication (ex: players at own console with no interaction)
- If $T_p = 0$, then turn based
- Between are MCGs which have some of both

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

Outline

- Introduction (done)
- Networking Resources (done)
- Distribution Concepts (done)
- Scalability (done)
- Security and Cheating (next)
- Conclusions

Security and 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)
  - *Dominance* – gain advantage (more)

Packet and Traffic Tampering

- *Reflex augmentation* - enhance cheater's reactions
  - Example: aiming proxy monitors opponents movement packets, when cheater fires, improve aim
- *Packet interception* – prevent some packets from reaching cheater
  - Example: suppress damage packets, so cheater is invulnerable
- *Packet replay* – repeat event over for added advantage
  - Example: multiple bullets or rockets if otherwise limited

Preventing Packet Tampering

- Cheaters figure out by changing bytes and observing effects
  - Prevent by MD5 checksums (fast, public)
- Still cheaters can:
  - Reverse engineer checksums
  - Attack with packet replay
- So:
  - Encrypt packets
  - Add sequence numbers (or encoded sequence numbers) to prevent replay

Information Exposure

- Allows cheater to gain access to replicated, hidden game data (i.e. status of other players)
  - Passive, since does not alter traffic
  - Example: defeat "fog of war" in RTS, 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 (example: attack enemy could not have seen)
    - Harder in replicated system, but can still share

Design Defects

- If clients trust each other, then if client is replaced and exaggerates cheater effects, others will go along
  - Can have checksums on client binaries
  - Better to have trusted server that puts into play client actions (centralized server)
- Distribution may be the source of unexpected behavior
  - Features only evident upon high load (say, latency compensation technique)
  - Example: Madden Football

Conclusion

- Overview of problems with MCGs
- Connection to other distributed systems
  - Networking resources
  - Distribution architectures
  - Scalability
  - Security

Future Work

- Other distributed systems solutions
- Cryptography
- Practitioners should be encouraged to participate