Networking Multiplayer Games

EECS 494 Fall 2005

Sugih Jamin
jamin@eecs.umich.edu

Multiplayer Games

Why multiplayer games?

- humans are better at most strategy than current AIs
- humans are less predictable
- can play *with* people, communicate in natural language
- add social aspect to computer games
- provides larger environments to play in, with more characters
- make money as a professional game player

People Online

by Web Site Optimization, LLC

http://www.websiteoptimization.com/bw/

Two Types of Multiplayer Games

Head-to-head death-match:

- fast-pace, intense interaction/combat
- no persistent state
- players form ad-hoc, short-lived sessions
- any client can be a server
- requires matchmaking service:
  built-in lobby or use GameSpy
- examples: X/NetTrek (1980s, simulation), Doom (1990s, FPS),
  Counter-Strike, StarCraft, AoE, etc. (RTS-combat)

Persistent-world, massively multiplayer online game (MMOG)

MMOG

Most MMOGs are MMORPGs:

- server(s) keep persistent states, players can drop in anytime
- traditionally emphasize social interaction (less combat, but changing)
- in the beginning: MUD/MOO (1978, text-based)
- first commercial titles: Meridian 59 (c. 1996) and others, together had $\leq 30,000$ players

Sugih Jamin (jamin@eecs.umich.edu)

MMORPGs

- Ultima Online (Origin Systems/EA, gold Sept. 27, 97):
  - o isometric view
  - o took 3 years to developed
  - o $> 100,000$ players in 1998 240,000 players in 2001, 225,000 in Apr. 2003
- Everquest (Verant/Sony, gold Mar. 16, 1999):
  - o first non-wireframe 3D entry,
  - o 300,000 players in 2000, 430,000 in 2002
  - o total revenue: $4 mil/month (BW, 11/9/01)

Sugih Jamin (jamin@eecs.umich.edu)

Most Popular MMORPG: NCSoft's *Lineage* and *Lineage II*

S. Korea's (Sept. 1998)

- 4 million players in 2003, 110,000 concurrent players!
- Lineage II (3D) developed by UO's/Destination Games' Richard Garriott (released Oct. 1, 2003)
- in the first 4 days: 130,000 players, 90,000 concurrent

Population of S. Korea: 50 million

Population of Seoul: 10 million

Sugih Jamin (jamin@eecs.umich.edu)

An Analysis of MMOG Subscription Growth

by Bruce S. Woodcock

http://www.mmogchart.com/

Networking in Games

Differentiate between in-game networking and backend infrastructure

Backend infrastructure (see Yahn Bernier's talk at GDC2000 rtsp://media.cmpnet.com/gamasutra/bernier.rm):

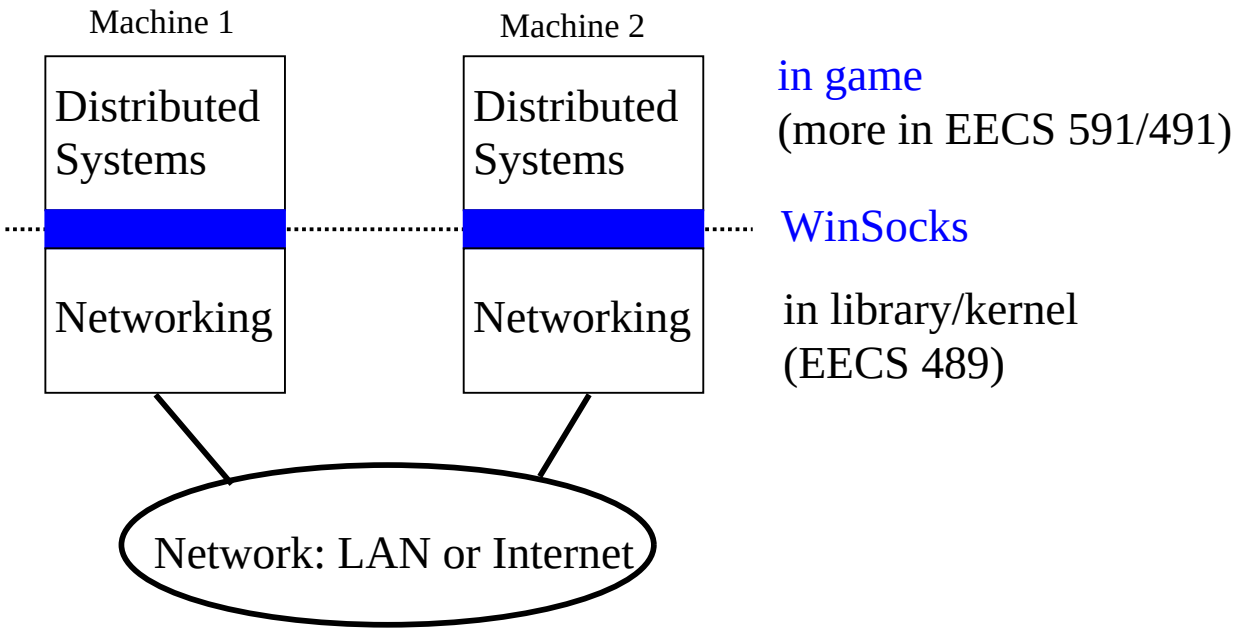
- lobby where gamers meet
- authentication and CD key checking
- accounting and billing
- ranking and ladder
- reputation and black list
- buddy lists, clans, and tournaments
- mods and patches management
- virtual economy
- beware of DDoS

Issues: scalability, adapting to failure, security

Networking in Games

In-game networking:

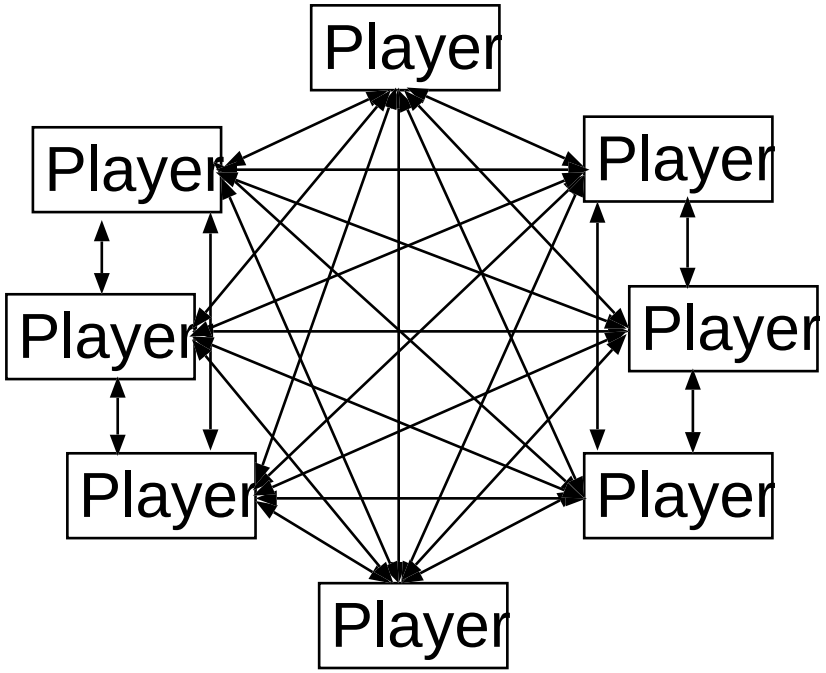
- networking topology: client-server vs. peer-to-peer
- computing model: distributed object vs. message passing
- which protocol to use? tcp, udp, reliable udp
- bandwidth limitation
- latency limitation
- consistency
- cheat proofing
- socket programming



Peer-to-Peer

Peer-to-peer with $O(N^2)$ unicast connections:

- each player is connected directly to all other players
- each player simulates the whole world
- advantages: reduced latency, no single point of failure
- disadvantages: easier to cheat, not scalable: each client must send and receive N-1 messages
- used in Age of Empire



Client-server

Two flavors:

- *ad-hoc* servers: death match
- dedicated servers: MMORG

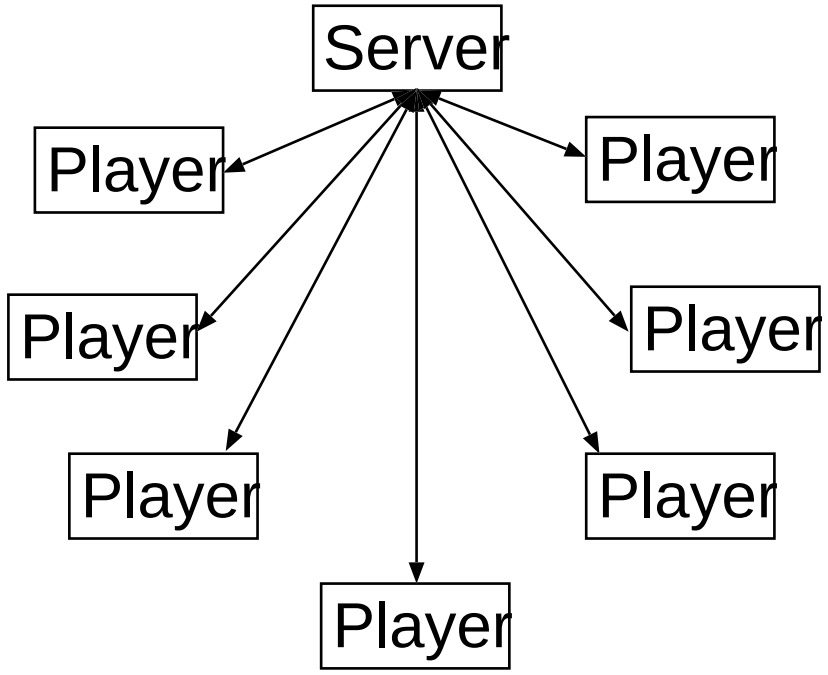
Two types of clients:

- clients simulate world, server has authoritative state: allows for client-side dead reckoning (QuakeIII/Half-Life).
- clients for I/O, all simulations at server: useful for thin clients, e.g. cell phones, and persistent-world MMOG.

Client-server

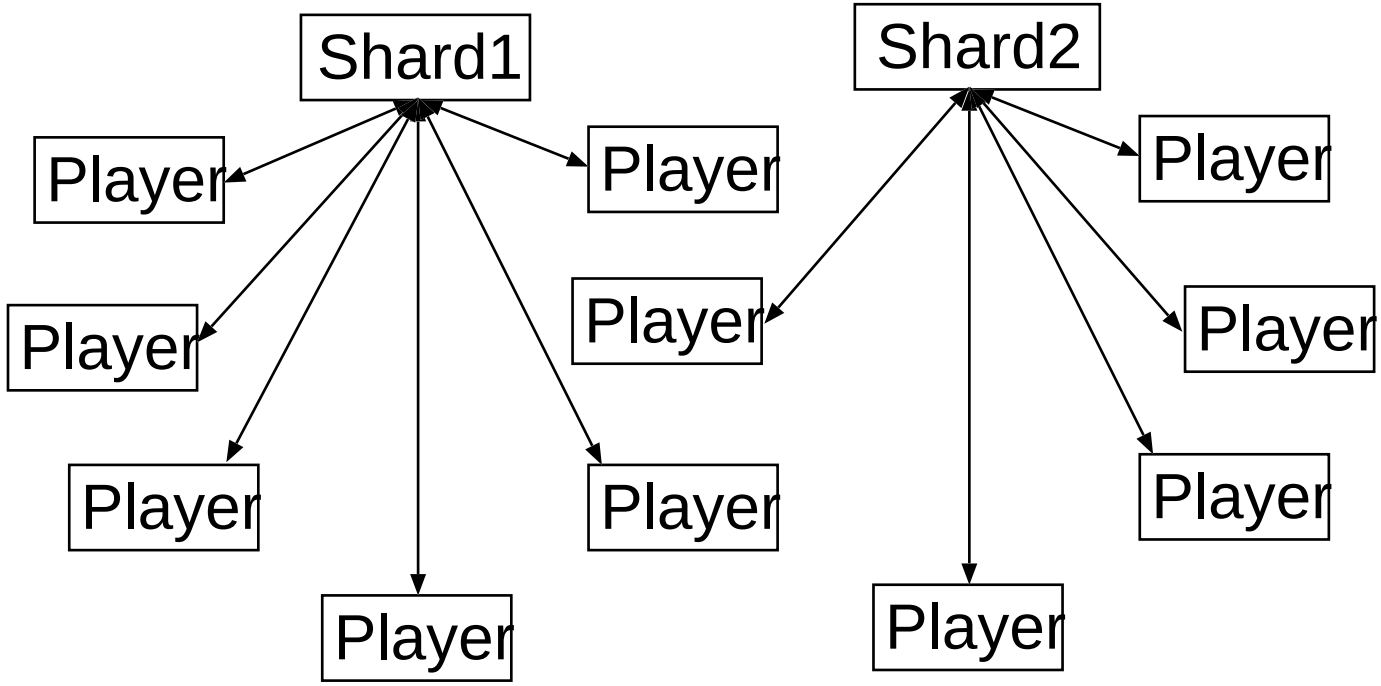
Client-server:

- advantages: each client sends only to server, server can aggregate moves
- advantages (dedicated servers): cheat-proofing, server can be better provisioned, persistent states (for MMOG)
- disadvantages: longer delay, server bottleneck, single point of failure, needs server management



MMOG Server Architecture 1

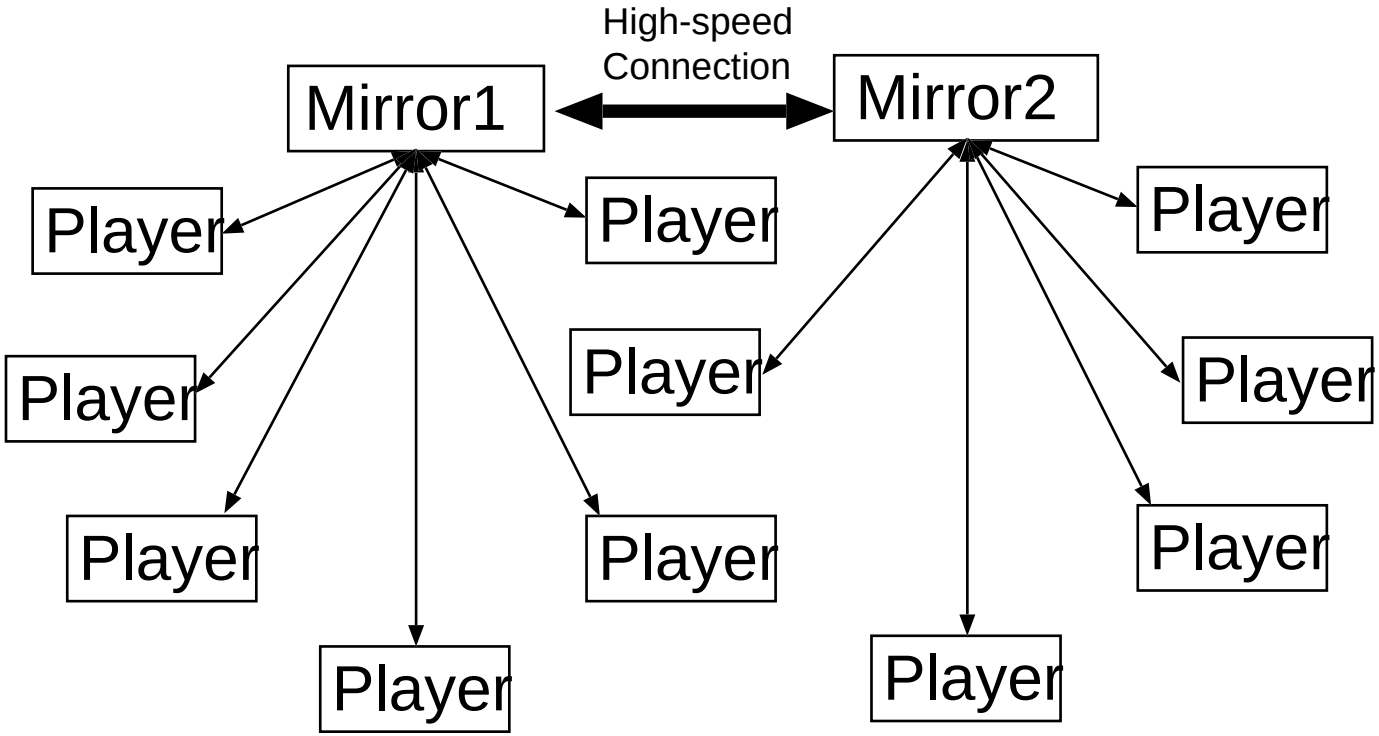
The world replicated at each server (shard); each shard contains an independent world; players go to specific shard:



Most MMORPG

MMOG Server Architecture 2

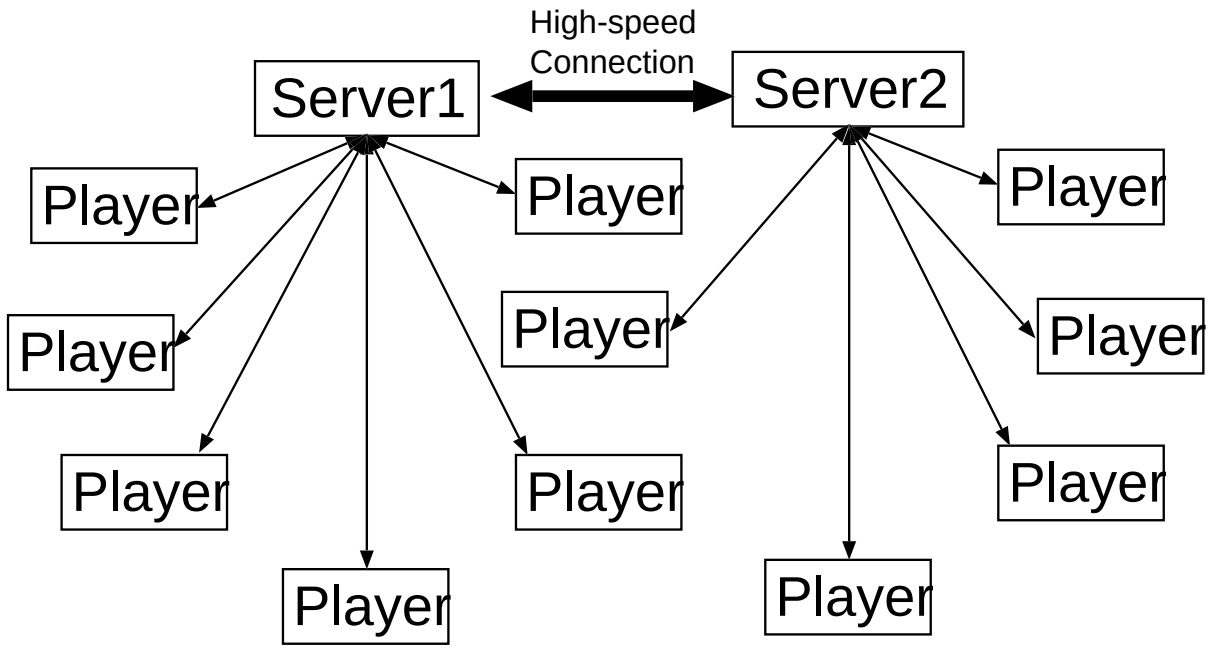
The world replicated at each server (mirror); all the worlds are synchronized; players see everyone across all mirrors:



Mirrors must be kept consistent

MMOG Server Architecture 3

The world is split up into regions, each region is hosted by a different server:



Example: *Second Life* from Linden Lab

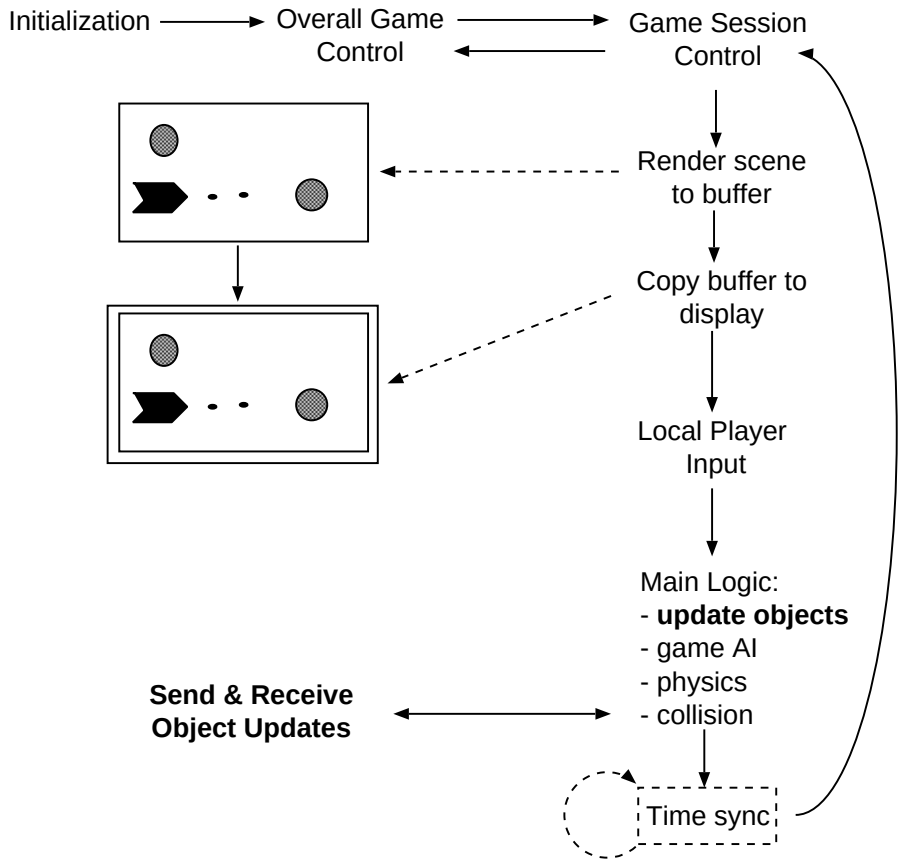
Servers must be kept consistent

Distributed Computing Model

Usually your game company will have its preferred computing model and would provide high-level libraries to implement the model

Distributed objects:

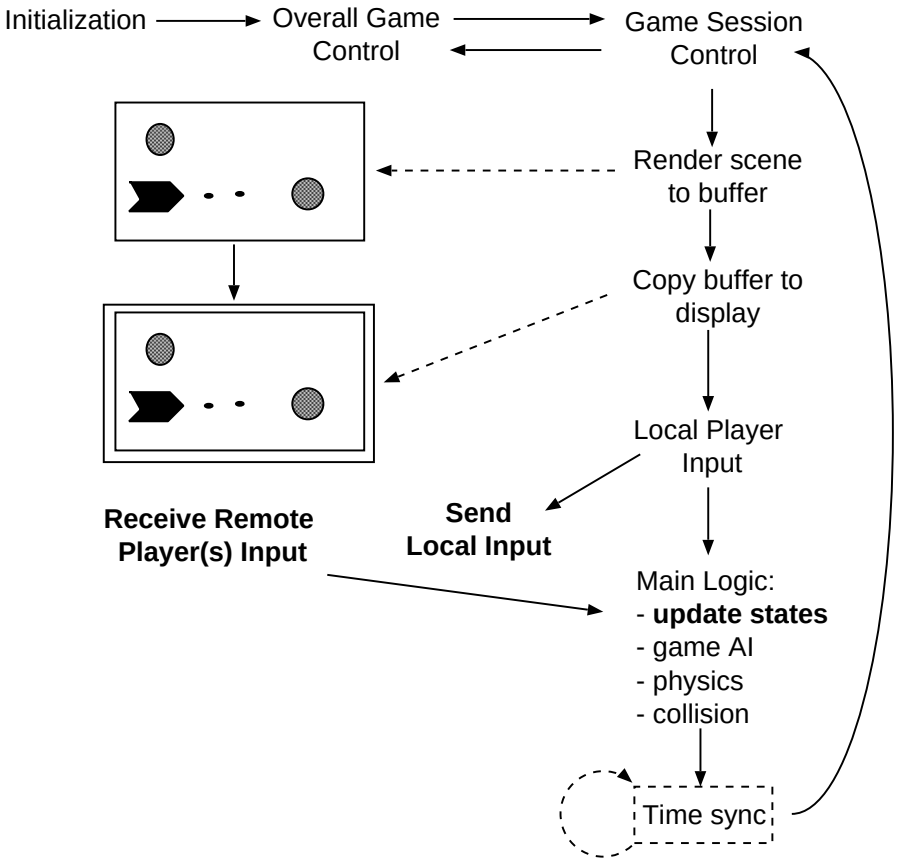
- characters and environment maintained as objects
- player inputs are applied to objects (at server)
- changes to objects propagated to all players at end of game loop
- object update usually implemented as one or more library calls



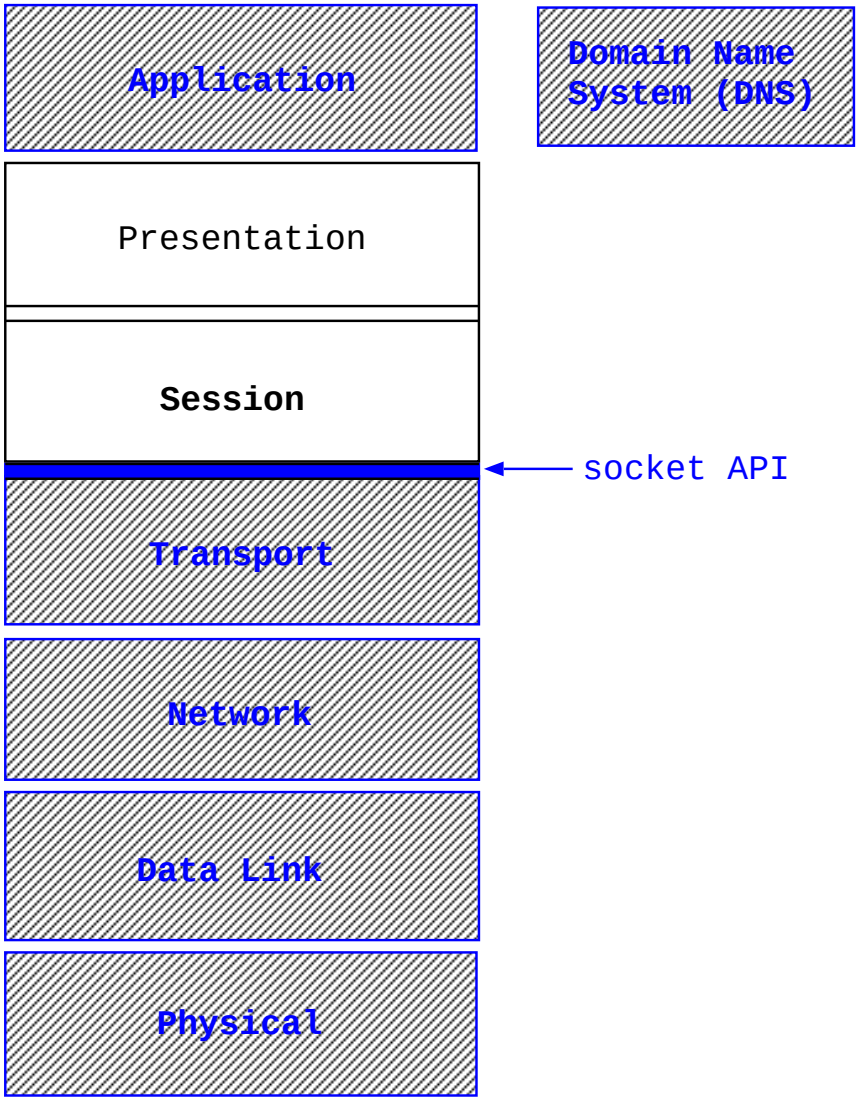
Distributed Computing Model

Message passing:

- player inputs (either button pushes or higher-level movements) are sent to other players (or server)
- all players update their own game state
- or server updates the global game state and send it out to all players



Which Protocol to Use? Protocol Layers



TCP vs. UDP

IP routes packet from source to destination,
max IP packet size is 64 KB, may be fragmented

What TCP (Transmission Control Protocol) gives you:

- reliable delivery
- retransmission and reordering
- congestion control

What UDP (User Datagram Protocol) gives you:

- unreliable delivery
- no retransmission, packets not ACKnowleged, no reordering
- no congestion control
- so, more or less, plain IP service

Which Protocol to Use?

Game requirements:

- late packets may not be useful anymore
- lost information can sometimes be interpolated
- but loss statistics may be useful

Use UDP in game:

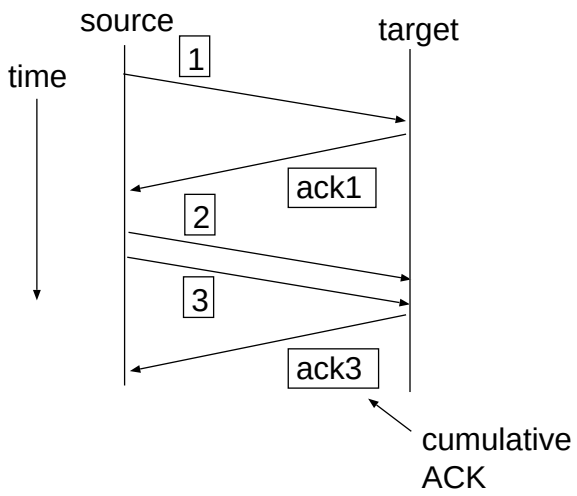
- can prioritize data
- can perform reliability if needed
- can filter out redundant data
- use soft-state
- send absolute values, not deltas
- or if deltas are used, send “baseline” data periodically
- must do congestion control if sending large amount of data

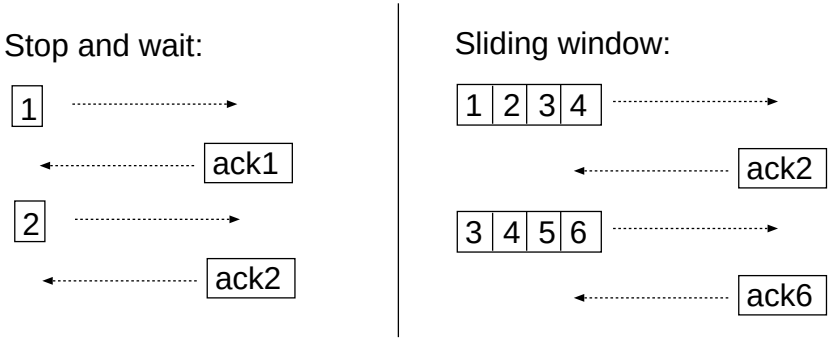
Reliable UDP

UDP doesn't provide reliability, write your own reliable udp for moves that **must** be reliable, e.g., snipper shots

Desirable features:

- **error control**: do checksum
- **ordering:** use sequence #
- **reliability:** acknowledge packet (use cumulative ACK), retransmit if not ACKed, timeout value a function of average rtt (round-trip time)
- **flow control:** don't send more than the target can handle; use stop-and-wait or sliding-window





Bandwidth Limitation

What is bandwidth?

What information is sent?

- depends on your computing model, distributed object or message passing
- game state: coordinates, status, action, facing, damage
- user keystrokes
- commands/moves
  For AoE: 1 every 1.5-2 sec, up to 3-4 commands/sec during battles (but some of these are redundant and can be filtered out)

Current lower limit assumed: 56 Kbps

Bandwidth Limitation (cont)

Bandwidth requirement has been HIGHLY optimized
Even with audio chat, takes up at most 8 Kbps

So, bandwidth is not a big issue (but note the asymmetric nature: at 8 Kbps, you can only support a total of 4 players on a 28.8 Kbps modem)

Must be continually vigilant against bloat

HOWEVER, with player-created objects and worlds, bandwidth becomes an issue again: use streaming, levels of details, and pre-fetching

Latency Limitation

How is latency different from bandwidth?

Latency:

- RTS: $\leq$ 250 ms not noticable, 250-500 ms playable, $>$ 500 ms noticable
- FPS: $\leq$ 150 ms preferred
- Car racing: $<$ 100 ms preferred, 100-200 ms sluggish, $\geq$ 500 ms, car out of control
- players' expectation can adapt to latency
- it is better to be slow but smooth than to be jittery
- don't rely on DirectPlay—at least test for its limitations