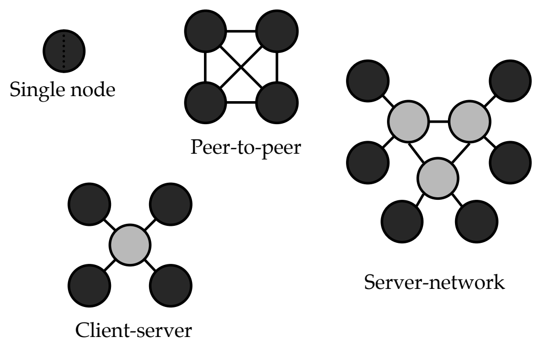
§8.2 Logical Platform

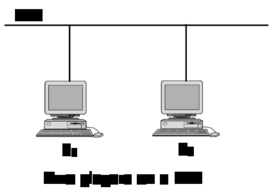
- communication architecture
  - peer-to-peer
  - client-server
  - server-network
- data and control architecture
  - centralized
  - replicated
  - distributed

Communication Architecture



Communication Architecture (cont'd)

- Logical connections
  - how the messages flow
- Physical connections
  - the wires between the computers
  - the limiting factor in communication architecture design



Example: How Many Players Can We Put into a Two-Player LAN?

- Distributed Interactive Simulation (DIS) protocol data unit (PDU): 144 bytes (1,152 bits)
- Graphics: 30 frames/second
- PDU rates
  - aircraft 12 PDU/second
  - ground vehicle 5 PDU/second
  - weapon firing 3 PDU/second
  - fully articulated human 30 PDU/second
- Bandwidth
  - Ethernet LAN 10 Mbps
  - modems 56 Kbps
- Assumptions:
  - sufficient processor power
  - no other network usage
  - a mix of player types

⇒LAN: 8,680 packets/second
fully articulated humans + firing = 263 humans
aircrafts + firing = 578 aircrafts
ground vehicles + firing = 1,085 vehicles

- Typical NPSNET-IV DIS battle
  - limits to 300 players on a LAN
  - processor and network limitations

Example (cont'd)

⇒Modem: 48 packets/second
fully articulated humans + firing = 1 human
aircrafts + firing = 3 aircrafts
ground vehicles + firing = 6 vehicles

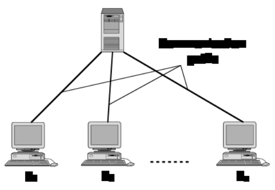
- Redesign packets
  - size 22%, 32 bytes

⇒Modem: 218 packets/second
fully articulated humans + firing = 7 human
aircrafts + firing = 14 aircrafts
ground vehicles + firing = 27 vehicles

- In a two-player game on a LAN, the protocol selection (TCP, UDP, broadcast,...) hardly matters
- As the number of live or autonomous players increase an efficient architecture becomes more important

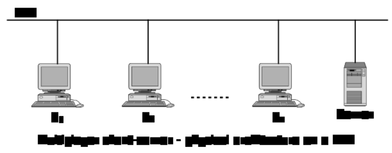
Multiplayer Client-Server Systems: Logical Architecture

- Client-server system
  - each player sends packets to other players via a server
- Server slows down the message delivery
- Benefits of having a server
  - no need to send all packets to all players
  - compress multiple packets to a single packet
  - smooth out the packet flow
  - reliable communication without the overhead of a fully connected game
  - administration

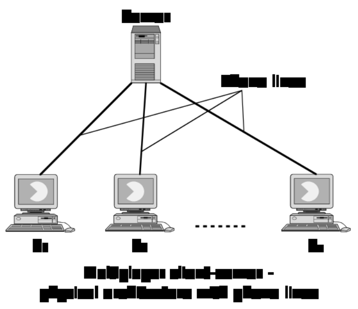


Multiplayer Client-Server Systems: Physical Architecture (on a LAN)

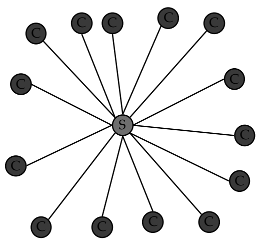
- All messages in the same wire
- Server has to provide some added-value function
  - collecting data
  - compressing and redistributing information
  - additional computation



Physical Architecture Can Match the Logical Architecture



Traditional Client-Server

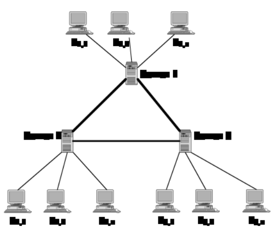


- Server may act as
  - broadcast reflector
  - filtering reflector
  - packet aggregation server
- Scalability problems
  - all traffic goes through the server

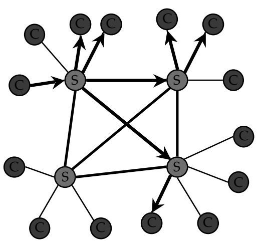
⇒Server-network architecture

Multiplayer Server-Network Architecture

- Players can locate in the same place in the game world, but reside on different servers
  - real world ≠ game world
- Server-to-server connections transmit the world state information
  - WAN, LAN
- Each server serves a number of client players
  - LAN, modem, cable modem
- Scalability

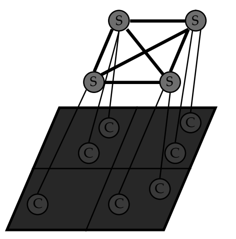


Partitioning Clients across Multiple Servers



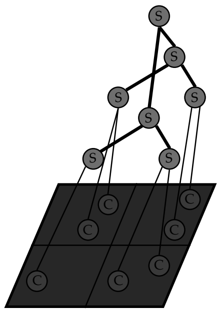
- The servers exchange control messages among themselves
  - inform the interests of their clients
- Reduces the workload on each server
- Incurs a greater latency
- The total processing and bandwidth requirements are greater

Partitioning the Game World across Multiple Servers



- Each server manages clients located within a certain region
- Client communicates with different serves as it moves
- Possibility to aggregate messages
- Eliminates a lot of network traffic
- Requires advanced configuration
- Is a region visible from another region?

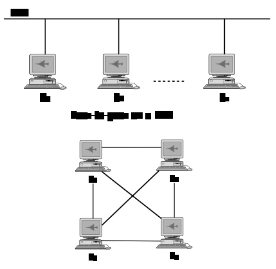
Server Hierarchies



- Servers themselves act as clients
- Packet from an upstream server:
  - deliver to the interested downstream clients
- Packet from a downstream client:
  - deliver to the interested downstream clients
  - if other regions are interested in the packet then deliver it to the upstream server

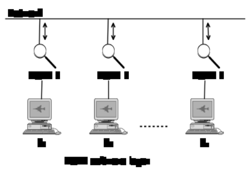
Peer-to-Peer Architectures

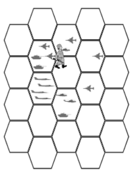
- In the *ideal* large-scale networked game design, avoid having servers at all
  - eventually we cannot scale out
  - a finite number of players
- Design goal
  - peer-to-peer communication
  - scalable within resources
- Peer-to-peer: communication goes directly from the sending player to the receiving player (or a set of them)



Peer-to-Peer with Multicast

- For a scalable multiplayer game on a LAN, use multicast
- To utilize multicast, assign packets to proper multicast groups
- Area-of-interest management
  - assign outgoing packets to the right groups
  - receive incoming packets to the appropriate multicast groups
  - keep track of available groups
  - even out stream information





Peer-Server Systems

- Peer-to-peer: minimizes latency, consumes bandwidth
- Client-server: effective aggregation and filtering, increases latency
- Hybrid peer-server:
  - over short-haul, high-bandwidth links: peer-to-peer
  - over long-haul, low-bandwidth links: client-server
- Each entity has own multicast group
- Well-connected hosts subscribe directly to a multicast group (peer-to-peer)
- Poorly-connected hosts subscribe to a *forwarding server*
- Forwarding server subscribes to the entities' multicast groups
  - aggregation, filtering

Data and Control Architectures

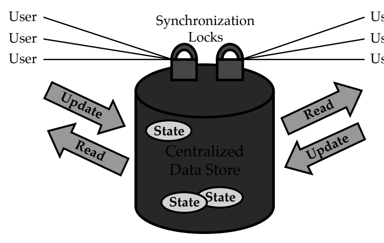
- Where does the data reside and how it can be updated?
- Centralized
  - one node holds a full copy of the data
- Replicated
  - all nodes hold a full copy of the data
- Distributed
  - one node holds a partial copy of the data
  - all nodes combined hold a full copy of the data
- Consistency vs. responsiveness

Requirements for Data and Control Architectures

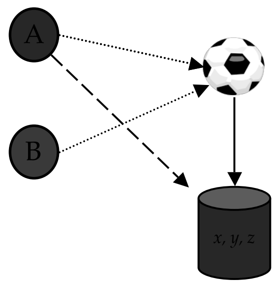
- Consistency: nodes should have the same view on the data
  - centralized: simple—one node binds them all!
  - replicated: hard—how to make sure that every replica gets updated?
  - distributed: quite simple—only one copy of the piece of data exists (but where?)
- Responsiveness: nodes should have a quick access to the data
  - centralized: hard—all updates must go through the centre node
  - replicated: simple—just do it!
  - distributed: quite simple—just do it (if data is in the local node) or send an update message (but to whom?)

Centralized Architecture

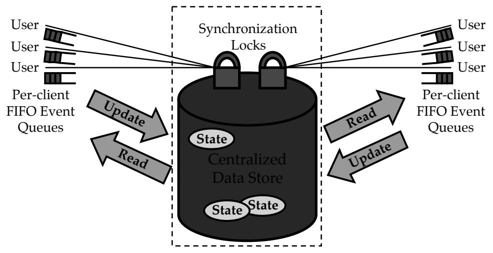
- Ensure that all nodes have identical information



Problem: Who's Got the Ball Now?



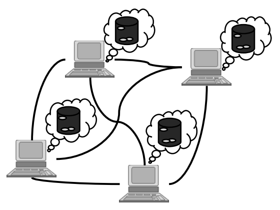
'Eventual' Consistency



Pull and Push

- The clients 'pull' information when they need it
  - make a request whenever data access is needed
  - problem: unnecessary delays, if the state data has not changed
- The server can 'push' the information to the clients whenever the state is updated
  - clients can maintain a local cache
  - problem: excessive traffic, if the clients are interested only a small subset of the overall data

Replicated Architecture

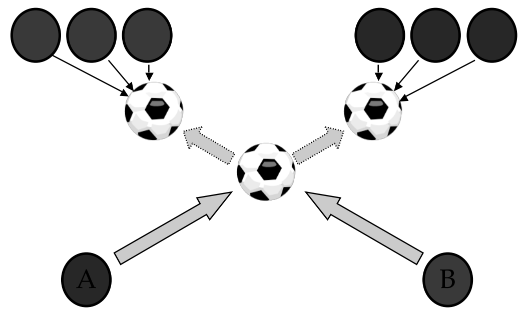


- Nodes exchange messages directly
  - ensure that all nodes receive updates
  - determine a common global ordering for updates
- No central host
- Every node has an identical view
- All state information is accessed from local node

Distributed Architecture

- State information is distributed among the participating players
  - who gets what?
  - what to do when a new player joins the game?
  - what to do when an existing player leaves the game?
- ⇒ Entity ownership

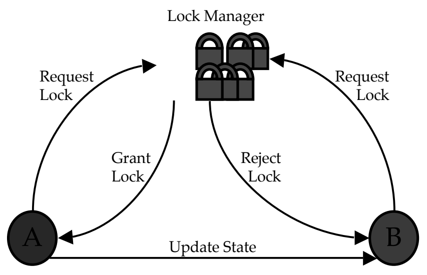
Problem: Who's Got the Ball Now? (Part II)



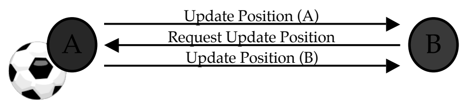
Entity Ownership

- Ensure that a shared state can only be updated by one node at a time
  - exactly one node has the ownership of the state
  - the owner periodically broadcasts the value of the state
- Typically player's own representation (avatar) is owned by that player
- Locks on other entities are managed by a lock manager server
  - clients query to obtain ownership and contact to release it
  - the server ensures that each entity has only one owner
  - the server owns the entity if no one else does
  - failure recovery

Lock Manager: Example

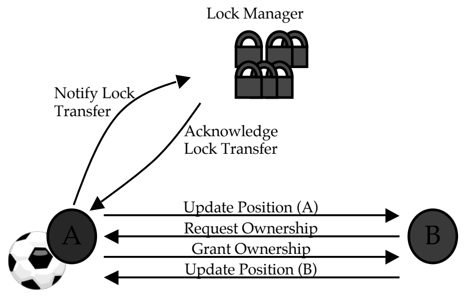


Proxy Update



- Non-owner sends an update request to the owner of the state
- The owner decides whether it accepts the update
- The owner serves as a proxy
- Generates an extra message on each non-owner update
- Suitable when non-owner updates are rare or many nodes want to update the state

Ownership Transfer



Ownership Transfer (cont'd)

- The lock manager has the lock information at all times
- If the node fails, the lock manager defines the current lock ownership state
- Lock ownership transfer incurs extra message overhead
- Suitable when a single node is going to make a series of updates and there is little contention among nodes wishing to make updates