Networking

- Networking is a crucial feature of many modern games
  - Used (until early 2000s) to be mainly PC
  - Halo: only over LAN
  - Halo 2 brought internet play to the consoles
  - Core feature of all consoles for two generations now
- Even if no network gameplay, often have social features:
  - 'Asynchronous multiplayer'
  - Online scoreboards
  - Ghosts, network challenges, cross-game rescue missions or advice (Shadow Of War, Dark Souls)

Challenges

- Has a huge impact on a game architecture
    - Complicates many aspects of the design
    - Very hard to retrofit after the fact
    - Massively multiplayer games have many technical and social issues
- Can have huge impact on company
    - May need to run servers
    - It’s not a feature that can be delegated to a couple of coders in the basement

Protocols

- TCP: Transmission Control Protocol
  - connection-oriented, byte stream service
  - guaranteed packet delivery
  - packets are received in-order
  - duplicate packets are discarded

- UDP: Universal Datagram Protocol
  - connectionless
  - unreliable delivery
  - out-of-order, duplicate packets possible

Protocols

- Most networking we care about is over IP
- There are a few environments (portables, phones) where you may use something else (e.g. LAN, Bluetooth)
- Games will generally have a fairly sophisticated layer on top of the base transmission protocol

TCP

- Pros
  - header compression
  - automatic segmenting
  - automatic congestion control
  - works with firewalls
  - reliable
- Cons
  - Deals with dropped/delayed packets in a non-timely manner
  - Connection will “block” till next portion of data stream is received

UDP

- Pros
  - Lower latency
  - No ACK send for packet receipt
- Cons
  - No compensation for dropped/missed packets
  - Generally need to write it yourself
  - Need to handle fragmenting yourself
  - May be blocked by network firewalls
  - In practice, can behave faultlessly on LAN, requiring you to test it over the internet or with debug code to simulate problems

Protocols

- UDP is generally used in games for bulk of communication
- TCP may be used for session management
  - Initial connection, matchmaking, meta-commands, etc.
  - Anywhere reliability is important, but speed isn’t
- Slow response to dropped packets rules out TCP
- Higher speed of UDP is nice too
- UDP requires a lot of extra work though
  - Ironically, you generally end up reimplementing a lot of TCP
  - Must be able to handle out of order and dropped packets
    - Simple approach: send all packets that have not been ACKed avery time
    - May need to compensate in game logic in some way

Architectures

- Several different approaches to take to the topology of the network
  - Client-server
  - Distributed
  - Peer-to-peer
  - Hybrid

Client-Server Architecture

- One machine responsible for the game state
- Clients send input, receive game state updates
- Game state synchronisation guaranteed
  - Because it’s frustrating when things go out-of-sync
  - Makes cheating a lot trickier
- Generally needs a good prediction system
- Server can be either dedicated or local to a client
- Workload concentrated
  - Good when server is dedicated
  - Bad when server is local
- Network traffic concentrated
  - May need fat pipe for server machine

Distributed Architecture

- Variation of dedicated client-server
- Multiple servers maintain game state
- No single server needs to know entire state
- Clients are handed off to servers as they move through the game
- Needed for massively multi-player games
  - Bandwidth / Computing needs are intense
  - Need to load balance
- Complex to develop, maintain
- Some client-server games without dedicated servers may be able to migrate who is the server

- Each player sends packets to all other machines
- No involvement of central server
- Traffic is uniform
- Load on players is uniform
- Less need for prediction
- Does not scale to large number of players
  - Client-server = 2n packets per update
  - Peer-to-peer = n^2 packets per update, O(n) probability that one of the peers is behind a strict NAT
- Vastly increases cheating potential

- Like regular P2P but all simulation/AI is deterministic
- Need only send controller input
- Cheating options greatly reduced
- Often used in sports games
- Writing a deterministic game is demanding
  - Even harder if cross-platform
- Hard to do when latency is high/variable and action is fast
  - Often need to wait for packets

Hybrid models

- Parts of the game follow P2P
  - Physics simulation
  - AI
- Other parts use client-server
  - Missions/quests
  - Player inventory
- Doesn’t quite address the cheating problem, but makes coding easier for less performance-critical systems
- Still requires some host migration
- Avoids much of the need to implement prediction

Game Protocols

- What should be in your game protocol?
  - Depends on the game (duh)
  - Client->server and server->client can be very different
    - Client sends single, relatively small input stream
    - Server sends updates for “all” entities
- Need a robust protocol
  - Sequenced, so that dropped packets can be detected and resent
  - Able to process packets out of order
    - So that dropped packets don't slow things down to much
  - Packets need to be small enough to pass through all router fragmenting (usually < 1500 bytes)
    - Compression (delta or just bit shaving)

What to Send

- Raw inputs with lockstep
  - Needs determinism
  - Very simple to implement (apart from determinism)
  - Spreads load / authority across clients
- Logical inputs / updates
  - Fixed set of operations (move to, fire weapon, take damage)
  - Can handle out of order / prediction easier
  - Limits extensibility (mods, engine reuse, etc.)
- General object replication
  - Replicate object variables
  - General-purpose RPC
  - Complicated to implement, but very powerful

Infrastructure

- Most online games require some infrastructure
  - Peer-to-peer games need matchmaking services
    - Often provided by platform (PSN, Xbox Live, Game Center)
  - Client-server games need:
    - Master servers (for matchmaking, if users run game servers)
    - Dedicated servers (for pay-to-play)
    - Authentication
- Need to make sure you have resources for this
  - Big pipes
  - Big servers
  - Reliability
  - A company that is in business

Performance

- Bandwidth (upper bounds)
  - LAN: 100/1000/10000 Mb/s
  - Cable/DSL: download 3-1000 Mb/s, upload 1-100 Mb/s
  - Mobile: 3-1000 Mb/s down, 1-100 Mb/s up
- Latency (lower bounds)
  - Wired LAN: 1 ms
  - Wireless LAN: 5 ms
  - Cell data: 50ms
  - Internet: 0 to 200+ ms
    - Depending on geographic location
- Crummy connections make lower bounds for bandwidth and upper bounds for latency much worse

Latency

- Latency is the primary problem for games
  - “Ping is king”
- Naive approach to game networking:
  - Clients send inputs for next update to server
  - Server waits for inputs from all clients
  - Update world
  - Sends new positions to all clients
  - Repeat
- Problem
  - Perceptual lag == ping of slowest client

Coping with Latency

- Minimal “good” networking architecture
  - Clients send updates at regular intervals
    - Time stamped
    - Not sequence dependent
  - Server sends position updates to clients at regular intervals
  - No one waits
  - Now perceptual lag is “only” your ping time
  - Rendering decoupled from simulation

Simple Prediction

- The client immediately moves based on local input
- Updates from the server get projected back to the past when received
  - Compare player state received from server to the historical (predicted) one cached locally
  - If the same, nothing to do, prediction was correct
  - Otherwise, re-predict with cached input sequence
- Removes the most annoying form of lag (movement of your character)
  - Other entities still lag

More Advanced Prediction

- Run all entity logic on client
  - But updates from the server can clobber it
  - Still get some teleporting from inability to guess input from human controlled entities
  - Still can get noticeable lag on instant hit weapons
- Problem: who gets the kill?
  - The Source engine deals with this by projecting client action into the past on the server
  - Good articles about it online

Cheating

Never trust the client.
Never put anything on the client.
The client is in the hands of the enemy.
Never ever ever forget this.

- Raph Koster

In every aggregation of people online, there is an irreducible proportion of jerks

- John Hanke's Law (cited by Mike Sellers)

Cheating

- Cheating can take the fun out of an online game
- Takes many forms:
    - Bots: programs that masquerade as players
        - Sometime an intentional part of the game (Quake 3, UT)
        - In MMORPGs people often have “bots” that do boring and repetitive, but profitable tasks unattended
    - Aiming proxies: programs to enhance the abilities of a player
    - Client-side mods: show all other players (visible or not), remove walls
        - If client is authoritative on anything, can be really damaging
    - Modified servers: give advantage to certain players
    - Unauthorised modification of world database (e.g. item duping – also spoils single player)

Combating Cheating

- Standard security techniques often don't work
    - We aren't dealing with someone trying to snoop or attack a hidden resource
    - Encryption is only vaguely useful
    - Security through obscurity is often a necessary evil
        - Hard to avoid keeping dangerous info on the client
- Make sure client isn't authoritative

Combating Cheating

- Central authentication
  - Needs to be built into pay services anyway (MMORPGs)
  - We used to use CD keys
  - Now most multiplayer is via a service like Steam or XBL so we have authentication that way
- Deal aggressively with offenders
  - This goes back to having a good infrastructure
  - Need tools for server ops (player-run servers or central)
  - Need to be able to kick, ban or relegate to “pool of shame”

Summary

- Design networking from the beginning
- Need a robust protocol
- Need to fight against lag
- Need to fight against cheating