Basic networking

- Computers communicate by sending *packets* of data to one another
  - Each packet contains a small amount of information
  - Larger data (e.g. files) are split into many packets
  - Packets are sent in order from the sender to the receiver
  - The receiver puts the packets back together as necessary

Local networks

- To connect to a network, a computer must have a *network interface*
  - Often called a *NIC* for *network interface card*
  - Network interfaces are connected by physical wires or radio waves
  - On most networks (e.g. *ethernet*), when one computer puts a packet on the wire, all other connected computers can see it
- Packets are addressed
  - Each NIC has a unique *MAC address* burned into it
  - Only the computer the packet is addressed to is supposed to keep it

Routing

- Once a local network gets too crowded, it slows down
  - Solution: hierarchical organization
- Network devices:
  - *Hub*: connects machines all together in one local network
  - *Switch*: like a hub, but is smart about sending packets only to the computers they're addressed to
  - *Router*: supports large-scale hierarchical networks

Internet protocol (IP)

- Internet Protocol (IP)
  - Used to *route* packets around the internet
  - Assigns a unique address to every computer on the internet
    - Well, not really anymore, but that's the abstraction
  - IPv4 address: 4 digits (e.g. 129.105.119.237)
  - First 3 digits typically specify the *subnet* (class-C)
    - Machines on the same subnet may or may not be physically together
- IP specifies a binary header for each packet
  - Concerned only with routing (delivery)
  - http://www.freesoft.org/CIE/Course/Section3/7.htm

Internet protocol (IP)

- Each router has multiple connections
  - Multiple paths it could send a packet
  - Routers talk to each other to establish which path a packet should follow
  - *Routing information protocol (RIP*) is the simplest
  - Only have to know where a subnet is, not each machine
  - Routers maintain tables of routing information and share with each other

Domain name service (DNS)

- IP addresses are hard to remember
  - Names are much easier
- The DNS system maps names to IP addresses
  - www.google.com, www.microsoft.com, www.utpa.edu
  - Domain name servers keep track of mappings
- DNS is hierarchical
  - UTPA has a DNS server for all our machines
  - Higher level DNS servers direct queries to our machines
- Use *nslookup* or *host* to perform DNS queries

TCP

- IP is only concerned with routing
  - Not concerned with the packet content
  - Not concerned with packet ordering
  - Doesn't guarantee delivery (no receipt)
- Transmission control protocol (TCP)
  - Works on top of IP
  - Creates a stateful abstraction (stream communication)
    - Reorders packets into proper sequence
    - Sends acknowledgement of packets
    - Requests re-send of missing packets
  - http://www.potaroo.net/papers/isoc/2004-07/tcp1.html

UDP

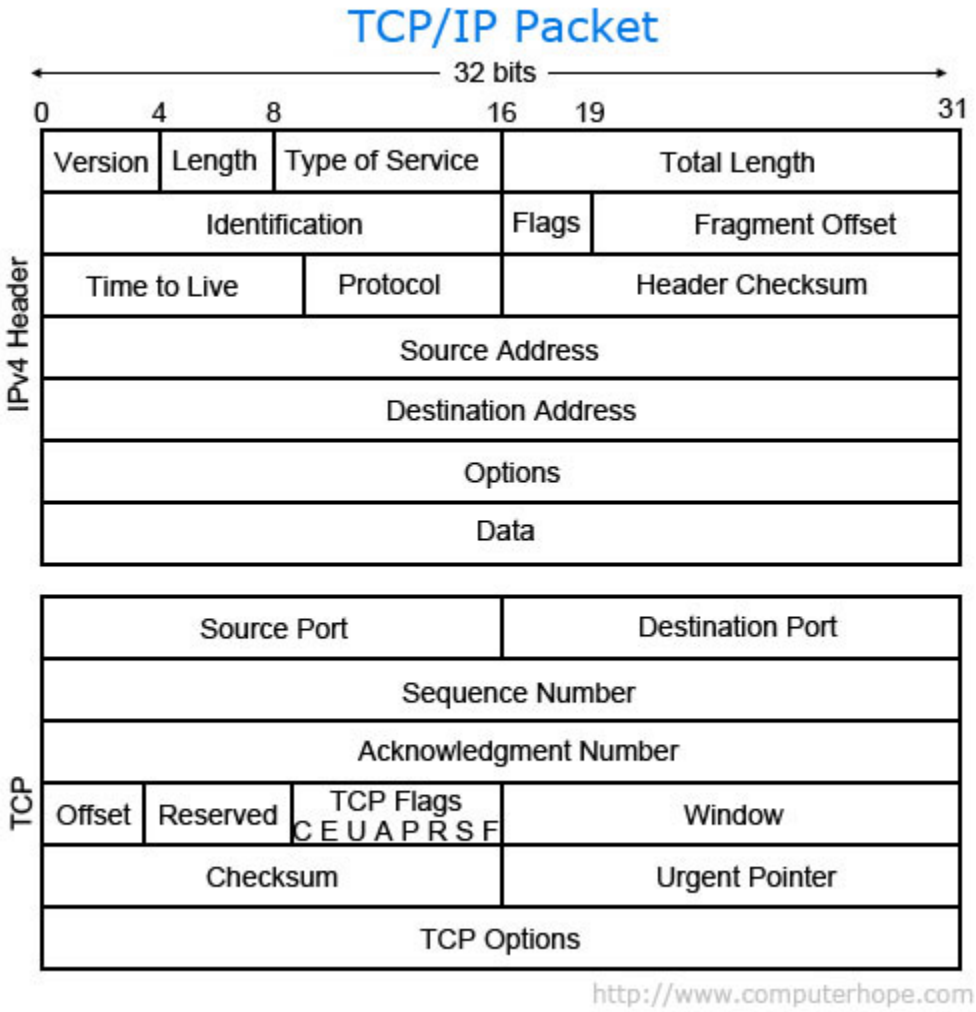
- User Datagram Protocol
  - Also works on top of IP, alternative to TCP
  - No state, no acknowledgement, no re-ordering, no re-sending, no flow control...
  - Just send packets to a port
    - Does carry a checksum for integrity verification
  - Unreliable, packets are lost (dropped)
    - Used when a missing packet doesn't matter as much as speed
    - Streaming video/audio
    - Game position updates (since they get replaced quickly)
    - *Not* for important, one-time in-game events

Topology

- Peer-to-peer
  - Every host responsible to update every other host
- Client/server
  - One host is designated server
  - Acts as communication hub (star topology)
- Dedicated server
  - Non-player machine
  - Acts as communication hub (star topology)

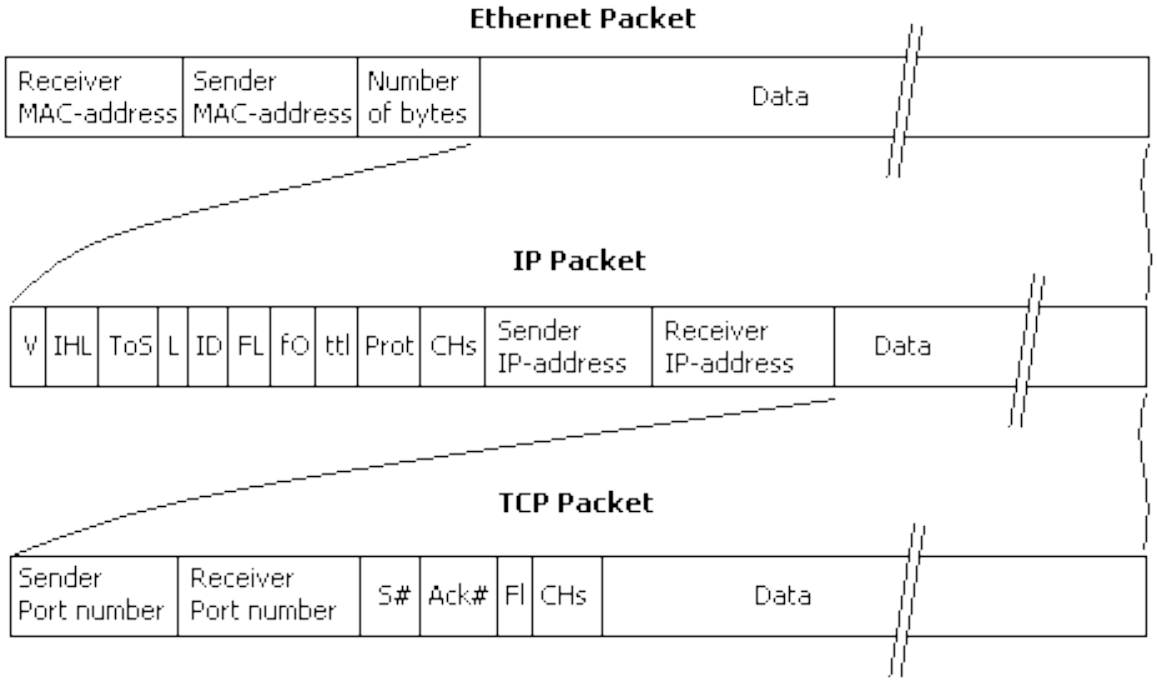
Network Packets

- Sequence of fixed-size fields
  - Way harder to work with than HTTP style key/value pairs
  - But far more efficient



Network Packets

- Fixed header fields pre-pended to (wrapped around) variable-sized data
  - Ethernet wraps IP, IP wraps TCP, TCP wraps application-specific *protocol*
  - Minimize unwrapping during transmission



Game Protocol

- Need to create a packet format
  - Fixed-size header fields, then variable data
    - E.g. Time or turn counter, client identifier, player identifier
  - Array of *operations*:
    - E.g. join, leave, select, fire
    - With operation-specific arguments

Naïve Networking

- Scenario: FPS
- Each host updates all other hosts (peer-to-peer)
  - Send current position, animation
  - Send “shoot” message, with target and hit/miss
- In game loop, pump network events
  - Just like you pump for user input events
  - Handle each packet by updating the world state

Problems!

- Network latency
  - Messages aren't sent instantaneously
  - Everyone else will see you move/shoot *after* you actually do it
    - Hundreds of milliseconds after
  - Assume 100ms latency (not terrible ping)
    - You shoot and hit Joe at time 5000
    - You're actually "hitting" him where he was at time 4900
    - The shoot/hit message gets to Joe at time 5100
    - He sees himself get hit based on where he was 200ms ago and where you were 100ms ago
  - Worse, cannot determine who shot first

Problems!

- Network congestion, message load
  - How often should you send a position update?
  - More frequent = more traffic, more load, more latency
  - Less frequent = less accurate, choppy
    - 20 fps can fool the human eye into seeing “motion”
    - Equivalent to update every 50ms

Problems!

- The client is in the hands of the enemy!
  - Never trust data from another player
- Many, many ways to cheat
  - Speedhack: lie about where you are
  - Maphack: make all players bright green and visible at all times
  - Aimbot: just say you always hit, or automatically calculate the correct aiming vector to guarantee a hit

Lockstep

- Used in classic RTS games
- Deterministic simulation
  - Always has the same outcome for the same inputs
- Only send inputs around
  - Everyone independently calculate outcome
  - Compare against each other to detect cheating
- Problem: one of the input parameters is time!
  - Latency makes time-of-input different on each machine
  - Solution: synchronize simulation steps
    - Everyone simulate one step, then wait for everyone else
    - Inputs are necessarily attached to the same step for everyone

Server Authoritative

- Lockstep is very inefficient
- Make the server authoritative
  - Clients just send inputs, server simulates and sends results
  - Used in FPS and MMOs
- Example:
  - Send forward key press to server
  - Wait...
  - Server simulates movement
  - Server sends position update to all clients
  - Problem: I just waited 200ms between keypress and action
  - Problem: heavy server load

Prediction and Reconciliation

- Responsiveness matters **a lot**
  - Waiting for the server to authorize your movement is not feasible
- Apply inputs immediately to client-side simulation
  - Player sees immediate action
    - Only an estimate of what happens, server must confirm
  - Send inputs to server, receive updated world state
    - When server packets are received, update local state to match
    - Server information is always correct
  - Problems: jitter, popping, events being undone
    - Smoothly interpolate from estimated position to real position

Dead Reckoning

- But what about everyone else?
- Extrapolation
  - Given known position, velocity, etc
  - Assume they keep the same input (e.g. holding down W)
  - Estimate their next position
  - When state arrives from the server, correct the local state
  - Much worse popping, unexpected hits/misses
- Event extrapolation
  - On input, start animation (e.g. spell casting)
  - Stretch out the animation until server response (hit/miss)
  - Avoids undoing actions and events

Dead Reckoning

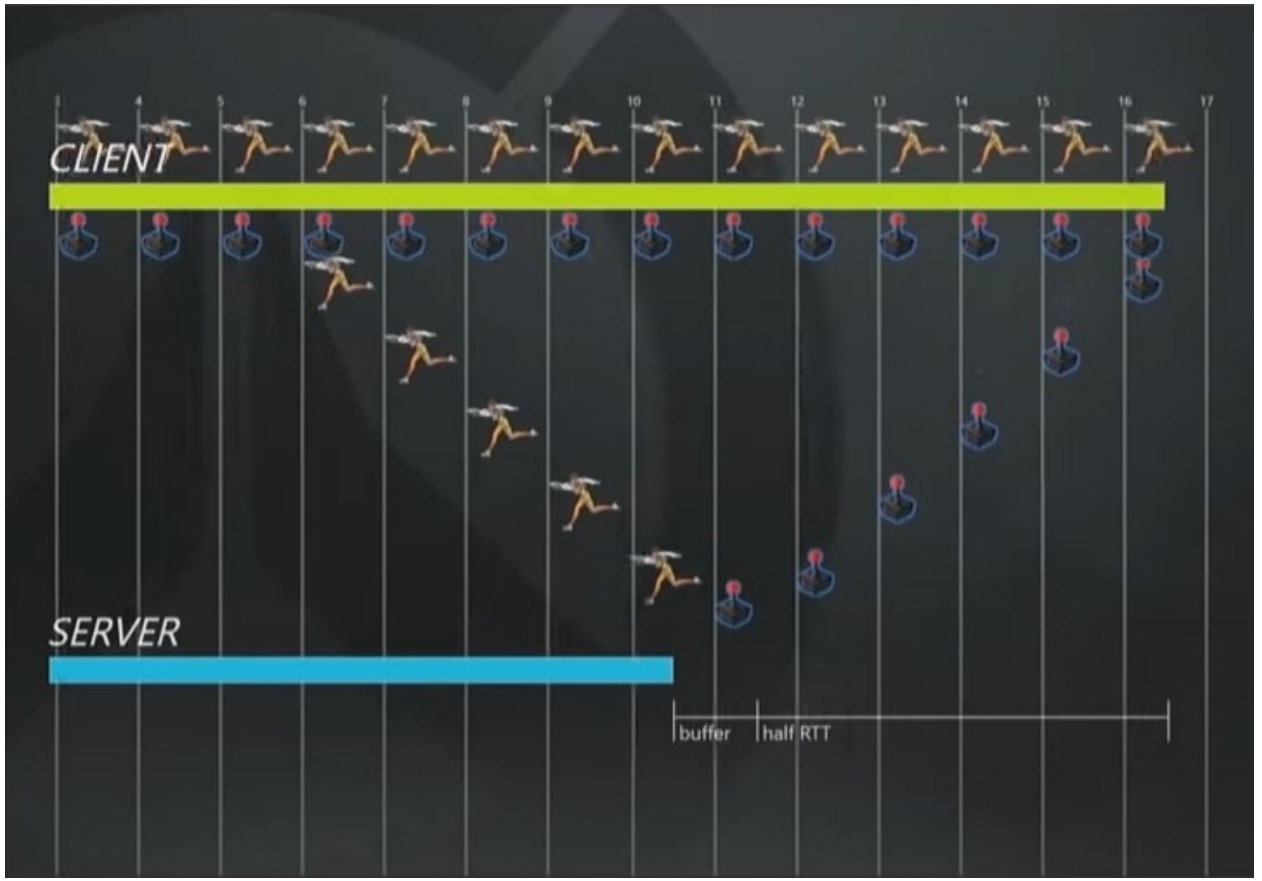
- Interpolation
  - Delay the simulation to work with only known state
  - Receive state data from server at 500ms and 600ms
  - Simulate other entities from 600ms to 700ms
    - Interpolate between known positions
    - Movement is real, just delayed 100ms
  - No popping if delay > lag
  - Player is in the present, enemies are in the past
    - Not as bad as it sounds for movement

Lag Compensation

- A perfect shot at 500ms
  - Is pointed at where the enemy was at 400ms
  - And will be checked on the server at 600ms
  - (They've probably moved by then and you miss)
- Instead, have the server check in the past
  - Check at 600ms
  - …where the player aimed at 500ms
  - …against where enemy was at 400ms
- Trade-off: delayed hits instead of phantom misses
  - Generally less objectionable

Rollback

- Time matters!
- Re-simulate whenever new information is received



Overwatch Gameplay Architecture and Netcode, Timothy Ford, Blizzard, GDC 2017
https://www.youtube.com/watch?v=W3aieHjyNvw&t=1738s

Partial Client Authority

- Clients send position, not input
  - Server does not have to simulate player movement
  - Used in MMOs to reduce load
  - Self-movement is not impacted by lag
  - Server must periodically check for speed hacking or teleporting
    - Random sampling
- Click targeting
  - Acquiring a target isn't an issue of aiming
  - Does not require server-side intersection tests

Server State Updates

- Standardized back around DOOM II/Unreal/Half-Life
  - Authoritative server model (either dedicated or a random player machine)
  - Prioritizing predictability
  - Send *complete* state snapshots at fixed tick (10-20Hz)
    - Relatively infrequent transmission
    - Delta-compress snapshots (only send what is different)
  - UDP, since dropped information is quickly replaced by the next snapshot

Simple Replication

- Unity GameObjects, UE Actors
  - Mark which attributes of which Entities (e.g. position of the player) should be included and “synchronized”
  - Set extrapolation/interpolation parameters to tune
  - Nice and simple abstraction, not suitable for competitive multiplayer w/o prediction, reconciliation, rollback, etc.
- Events/RPC
  - Used to send infrequent, reliable messages for important events (e.g. kills, pickups)

Further Reading

- Simple guide to modern competitive multiplayer netcode features
  - https://gabrielgambetta.com/client-server-game-architecture.html

- Canonical paper on RTS lockstep networking:
  - https://www.gamedeveloper.com/programming/1500-archers-on-a-28-8-network-programming-in-age-of-empires-and-beyond
- Half-Life networking (lag compensation)
  - https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization#Lag_Compensation
- Detailed description of the Doom III model:
  - http://mrelusive.com/publications/papers/The-DOOM-III-Network-Architecture.pdf