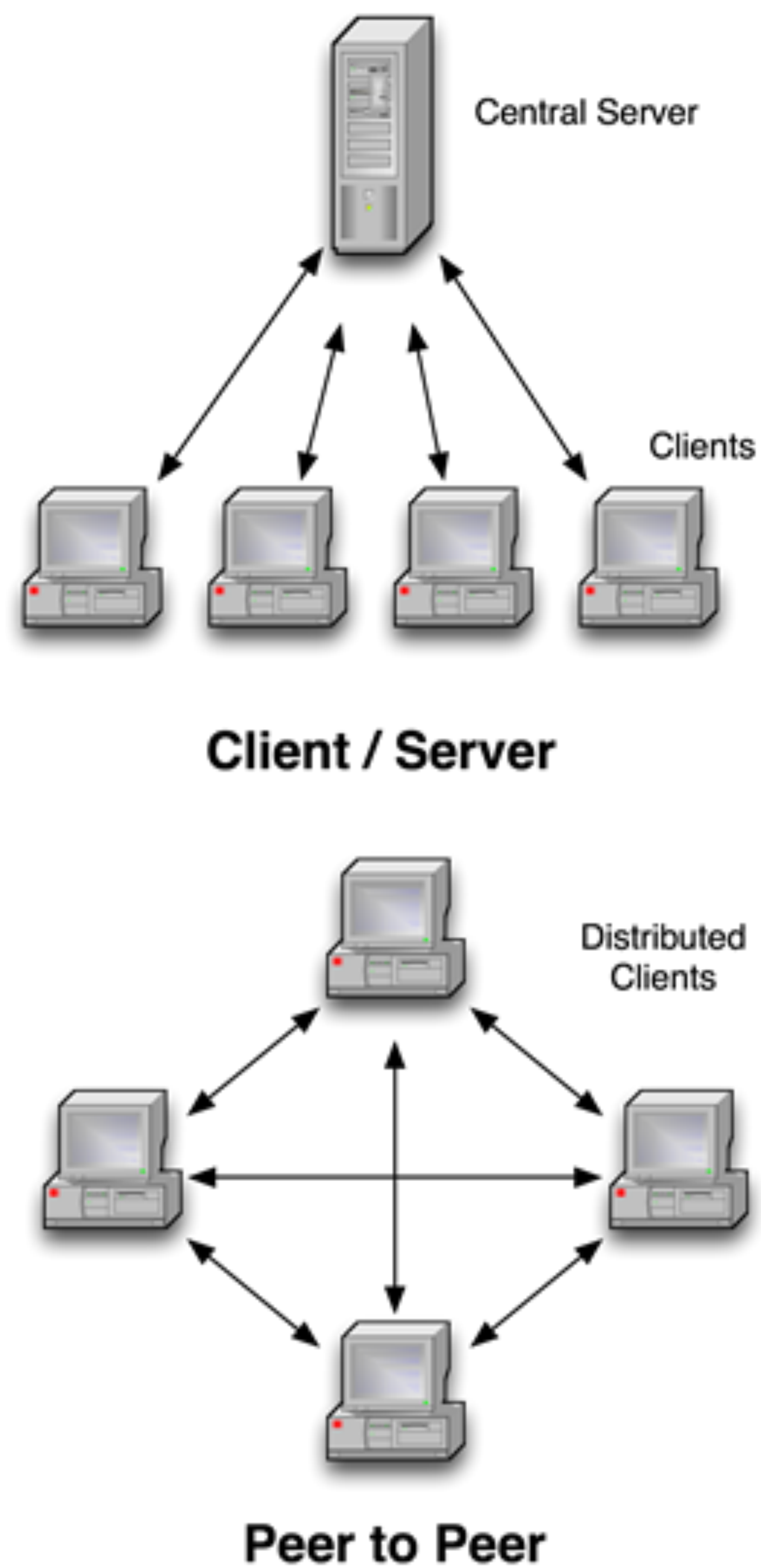
CS354R

DR SARAH ABRAHAM

NETWORKING OVERVIEW

NETWORKING FOR GAMES

- Networking required for multi-player games
- In **persistent** games, state remains regardless whether anyone is playing
- In **transient** games, state exists only while people are playing
- Client-server versus P2P models



NETWORKING CONCERNS

- Four primary concerns in building networked games:
  - Latency: How long does it take for state to be transmitted
  - Reliability: How often is data lost or corrupted
  - Bandwidth: How much data can be transmitted in a given time
  - Security: How is the game-play protected from tampering
- All of these considerations interact and require tradeoffs…

LATENCY IN GAMES

- Latency is the time between when the user acts and when they see the result
- Arguably most important aspect of a game network
  - Too much latency makes gameplay hard to understand (cannot associate cause to effect)
- Latency is **not** the same as bandwidth
  - A freeway has higher bandwidth than a country road, but the speed limit (latency) can be the same
- Excess bandwidth can reduce the variance in latency, but cannot reduce the minimum latency (queuing theory)

SOURCES OF LATENCY

- You cannot make any of these sources go away!
- You don't even have control over some of them!

- Frame rate latency
  - Data goes out/in from the network layer once per frame
  - User interaction only sampled once per frame
- Network protocol latency
  - Time for OS to put/get data on physical network
- Transmission latency
  - Time for data to be transmitted to the receiver
- Processing latency
  - Time for the server (or client) to compute response

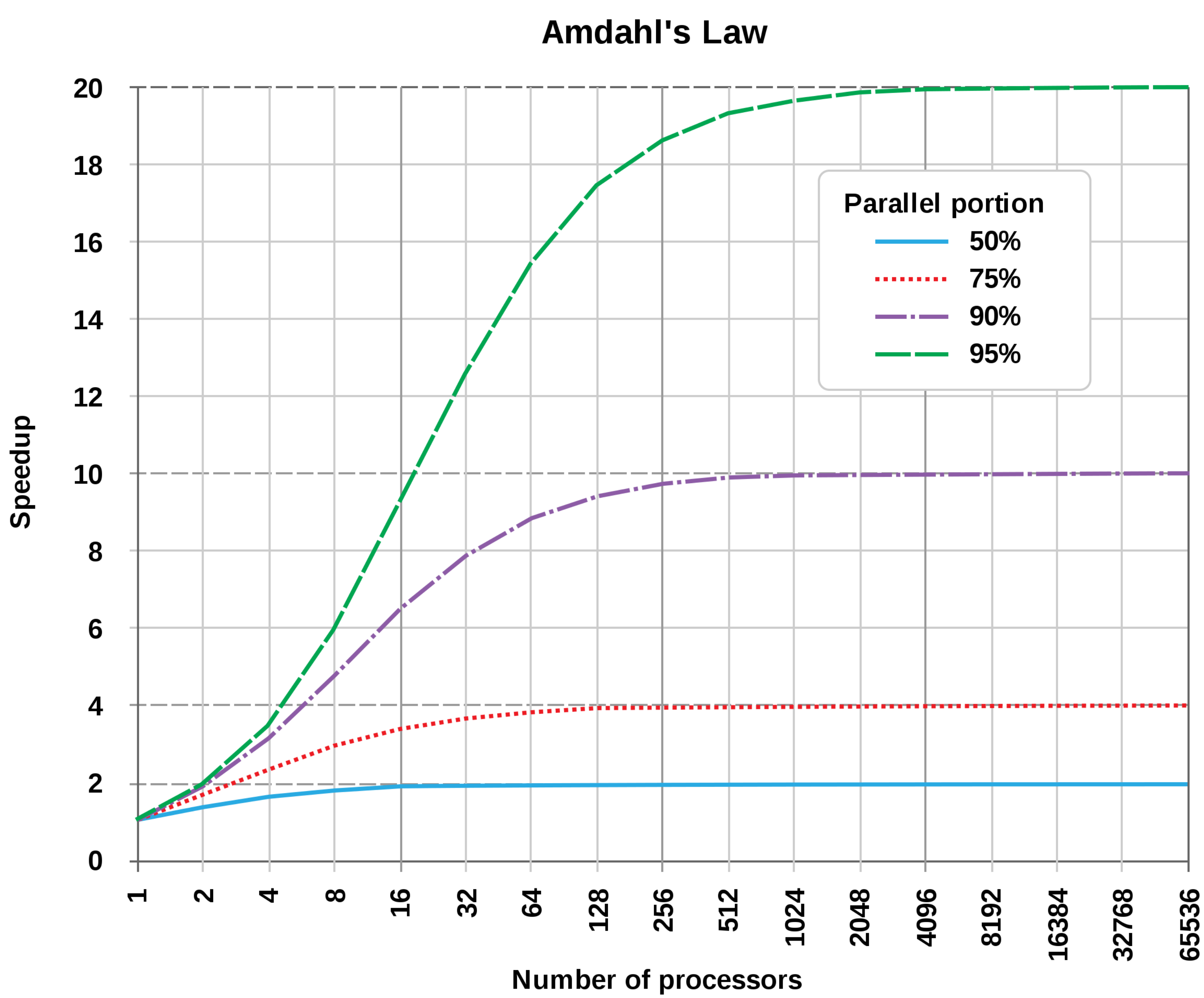
SIDE NOTE: AMDAHL'S LAW

- Formula for theoretical speed up in latency based on improved resources

$$S_{latency}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$

where s is speed up benefit and
p is portion of task benefitted

- Basically it says additional resources won't help performance if you have bottlenecks with fixed latency



REDUCING LATENCY

- Frame rate latency:
  - Increase the frame rate (faster graphics, AI, physics, etc)
- Network protocol latency:
  - Send less stuff
  - Switch to a protocol with lower latency
- Transmission latency:
  - Send less stuff
  - Upgrade your physical network
- Processing latency:
  - Make your server faster
  - Have more servers

BUT…

- The sad fact is, networking researchers and practitioners are almost never concerned with latency
  - Many (non-game) applications can handle higher latency
  - When have you heard a DSL/Cable ad promising lower latency?

WORKING WITH LATENCY

- Hide latency, rather than reduce it
- Any technique will introduce some form of error
  - Impossible to provide immediate, accurate information
- **Option 1:** Sacrifice game-play:
  - Deliberately introduce lag into local player's experience to allow time to deal with the network
- **Option 2:** Sacrifice accurate information:
  - Show approximate positions
  - Ignore the lag by showing "old" information about other players
  - Guess where the other players are now

LATENCY EXAMPLE: EVE ONLINE

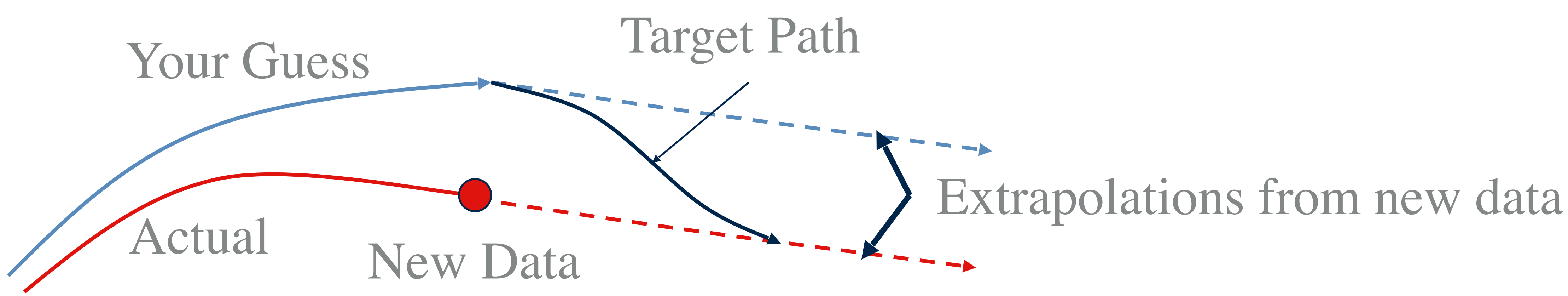
- https://youtu.be/TLqb-m1ZZUA?t=194
- https://io9.gizmodo.com/5980387/how-the-battle-of-asakai-became-one-of-the-largest-space-battles-in-video-game-history

DEAD RECKONING

- Use prediction to move objects even when their positions are not precisely known
  - Client maintains precise state for local objects
  - Client receives updates of the position of other players, along with velocity or acceleration information
  - Non-local positions extrapolated
- Within a client-server model, each player runs their own version of the game (replication), while the server maintains absolute authority
- Reduces the appearance of lag

FIXING EXTRAPOLATION ERRORS

- New position, velocity, and acceleration data for other players arrives
  - This doesn’t agree with their position in your local version
- Two options:
  - Jump to the correct position
  - Interpolate the two positions over some period
    - Path will never be exact, but will reasonably match



NETWORK RELIABILITY CONSIDERATIONS

- Does everything need to be completely reliable in games?
- Are all aspects of reliability equally important?

NETWORK RELIABILITY

- Protocol can attempt to ensure every packet is delivered
  - Incurs costs in terms of latency and bandwidth
- Other protocols try less hard to ensure delivery
  - Won't tell you if packets get lost
  - Latency and bandwidth requirements are lower

NETWORK RELIABILITY

- Other aspects of reliability:
  - **Error checking:** do the right bits arrive?
  - **Order consistency:** do things arrive in the order they were sent?

RELIABILITY REQUIREMENTS

- What information must be reliable?
  - Data that has no chance to recapture if it goes missing
    - Discrete changes in game state
    - Information about payments, joining, dropping etc
- What information does not need to be reliable?
  - Information that rapidly goes out of date
  - Information that is sent frequently
- Note that data that goes out of date quickly is sent more often
  - Big payoffs for reducing the cost of sending

INTERNET PROTOCOLS

- Only two internet protocols are widely deployed and useful for games:
  - **TCP/IP** (Transmission Control Protocol/Internet Protocol) is most commonly used
  - **UDP** (User Datagram Protocol) is also widely deployed and used
- Other protocols exist:
  - Proprietary standards
  - Broadcast and Multicast are standard protocols with some useful properties, but not widely deployed
  - If the ISPs don't provide it, you can't use it

TCP/IP OVERVIEW

- Advantages:
  - Guaranteed packet delivery
  - Ordered packet delivery
  - Packet checksum checking (some error checking)
  - Transmission flow control
- Disadvantages:
  - Point-to-point transport
  - Bandwidth and latency overhead
  - Packets may be delayed to preserve order
- Uses:
  - For data that must be reliable, or requires one of the other properties
  - Games that can tolerate latency

UDP OVERVIEW

- Advantages:
  - Packet-based (works with the Internet)
  - Low overhead in bandwidth and latency
  - Immediate delivery (no wait for ordering)
  - Point-to-point and point-to-multipoint connection
- Disadvantages:
  - No reliability guarantees
  - No ordering guarantees
  - Packets can be corrupted
  - Can cause problems with some firewalls
- Uses:
  - Data that is sent frequently and goes out of date quickly

CHOOSING A PROTOCOL

- Decide on the requirements and find the protocol to match
- Can use both protocols in the same game
- You can also design your own "protocol" by designing the contents of packets
  - Add cheat detection or error correction, for instance
  - Wrap protocol inside TCP/IP or UDP
  - Not actually a true protocol!

REDUCING BANDWIDTH DEMANDS

- Bandwidth is plentiful these days…
  - But it becomes an issue with large environments
- Smaller packets reduce both bandwidth and latency
  - Be smart about what you put in your payload and how!
- Dead reckoning reduces bandwidth demands by sending state less frequently
- What's another way to reduce bandwidth and latency?
  - Think open worlds/MMOs…

AREA OF INTEREST MANAGEMENT

- Area of interest management is the networking equivalent of visibility check
  - Only send data to the people who need it
- Doesn't work if network doesn't know where everyone is
- Area-of-interest schemes employed in client-server environments:
  - Server has complete information
  - Server decides who needs to receive what information
  - Only sends information to those who need it
- Two approaches: grid and aura methods

GRID AND AURA METHODS

- Grid methods break the world into a grid
  - Associate information with cells
  - Associate players with cells
  - Only send information to players in the same (or neighboring) cells
- Aura methods associates an aura with each piece of information
  - Only send information to players that intersect the aura
- Players need to find out all the information about an entered space regardless of when that information last changed
  - Why might this be tricky?

CONSIDER…

- A player opens a door in a multiplayer game
  - What needs to be replicated to the other players?
  - When should these things be replicated to other players?

SECURITY

- Basic rule of security: treat all clients as malicious and adversarial!
- The data clients send to server should not be trusted
  - P2P model inherently unsafe
  - In client-server model, server should verify all incoming packets from client
  - Limit entry-points and communication between server and client
- Remember:
  - Players might just be malicious…
  - But they could also be organized crime!

SECURITY EXAMPLE: DARK SOULS

- https://www.youtube.com/watch?v=9cF1DvOiiUA

SECURITY EXAMPLE: COUNTER STRIKE GO

- https://kotaku.com/top-counter-strike-players-caught-in-big-cheating-scand-1662810816

SECURITY CONSIDERATIONS

- What are some of the game-specific things we should be checking on server side to prevent malicious/cheating behaviors?

NETWORKING IN FIGHTING GAMES

- Fighting games have specific requirements in terms of networking
  - Work well as peer-to-peer connections
  - Send packets of input data to be processed on remote machine
- What makes networked fighting games hard?

PACKET LATENCY AND PROCESSING

- Packets take time to reach destination but contain frame-sensitive data
  - Input must be **timely and ordered** to correctly match behavior on sender's side
- Computers run at different speeds and may drop frames
  - Computer performance for one player impacts how actions are processed
- What do we need to ensure to have a good play experience?

CONSISTENCY OF VIEWS

- Both players must have a consistent view of the world with consistent frames in the world
- Two ways of working with this:
  - Input delay
  - Rollback

INPUT DELAY

- Delay both player inputs by same frame amount
  - Calculate frame delay based on players' ping
  - Only run input when input for that frame has been received
  - Send multiple frame inputs per frame packet to reduce waiting on specific frame data
- Pros:
  - Relatively simple and cheap to calculate
  - Ensures both players share same frame times
- Cons:
  - Feels terrible

ROLLBACK

- System predicts remote player's inputs and rolls back when new input is received
- Possible to combine rollback with input delay
  - Reduces teleporting and sudden state changes
- GGPO (Good Game Peace Out) is middleware solution for fighting game netcode created by a fighting game player
  - Uses rollback
  - Now licensed by most major fighting games

NETWORKING IN GODOT

- Godot allows for high-level networking through `MultiplayerPeer` interface
  - Inherits from `PacketPeer`, which performs serialization of packets
  - `ENetMultiplayerPeer` is high-level implementation
- Must associate networking object with the `SceneTree`
  - Based on how networking object is initialized, parts of the Scene Tree will either be a server or a client
  - Check if server or client with `is_network_server()`
  - Clients connected with unique id that can be retrieved through the `SceneTree`

SCENE INSTANCING

- Each player needs own scene object that is connected to the `SceneTree` of every other player
  - Load in self to local `SceneTree`
  - Load in remote players to local `SceneTree`
  - Can name remote players' scene nodes after their unique id
- Scene instancing in GDExtension follows a similar process

WHAT NEXT?

- Players have connected and are instanced to all other players – what next?

RPCS IN GDSCRIPT

- Remote Procedure Calls used to communicate between peers
  - RPCs used to call programs in a different address space
  - Address space can be another machine
  - Allows for calls to be the same locally and across the network
- Godot has RPC functionality built into `Node` objects through a highlevel multiplayer API
  - Uses the `@rpc` annotation in GDScript

```
@rpc([annotations])
func my_rpc_call():
```

- Call function as an rpc using Callable

```
my_rpc_call.rpc() //Calls on every peer
my_rpc_call.rpc_id() //Calls on a specific peer with id
```

RPCS IN GDEXTENSION

- Bind function that will be called using an RPC
  - `ClassDB::bind_method(D_METHOD("myFunction"), &MyClass::myFunction);`
- Set annotations for RPC using `rpc_config` before making RPC call
  - `rpc_config(“myFunction”, myConfigDictionary);`
  - `myConfigDictionary` will contain all necessary key-value pairs for RPC call
- Use `rpc` call to send bound function with arguments
  - `rpc(“myFunction”, myFunctionArguments);`

RPC ANNOTATIONS

- Godot provides modes to specify how methods are called by RPCs (by default, methods are disabled for RPC calls)
- Basic parameters for RPC annotations/config are:
  - Mode:
    - Authority (only multiplayer authority can call remotely)
    - Any Peer (clients allowed to call remotely)
  - Sync:
    - Call Remote (not called by local peer)
    - Call Local (can be called by local peer)
  - Transfer Mode:
    - Unreliable (no acknowledgement, can arrive out of order)
    - Reliable (resent until packet acknowledged, must arrive in order)
    - Unreliable Ordered (no acknowledgement but packets received in order sent)

MODE CONFIGURATION

- GDScript uses text strings, GDExtension/plugins use RPCMode enum
  - Disabled: by default (RPC_MODE_DISABLED)
    - RPC calls not accepted by method or property
  - Authority: “authority” (RPC_MODE_AUTHORITY)
    - Remote calls only accepted by the multiplayer authority (server by default)
  - Any Peer: “any_peer” (RPC_MODE_ANY_PEER)
    - Calls accepted from all remote peers

SYNCHRONIZATION CONFIGURATION

- GDScript uses text strings, GDExtension/plugins uses booleans
  - Remote call: “call_remote”
    - Function will be called on all remote peers
  - Local call: “call_local”
    - Function called on the local peer

TRANSFER CONFIGURATION

- GDScript uses text strings, GDExtension/plugins use TransferMode enum
  - Reliable: “reliable” (TRANSFER_MODE_RELIABLE)
    - Requires resending if out of order or packets are lost. Use sparingly
  - Unreliable: “unreliable” (TRANSFER_MODE_UNRELIABLE)
    - No acknowledgement or resend attempts. Use for non-critical data
  - Unreliable Ordered: “unreliable_ordered” (TRANSFER_MODE_UNRELIABLE_ORDERED)
    - Receives packets in order by ignoring late packets. Useful for positional data

NOTE: MULTIPLAYER PEER

- Mid-level object that provides an interface to multiple C++ implementations
- Extends PacketPeer to hand serialization, setting peers and transfer modes, and detecting peer connects/disconnects
- Godot provides three implementations:
  - ENetMultiplayerPeer (ENet)
  - WebRTCMultiplayerPeer (WebRTC)
  - WebSocketPeer (WebSocket)
- High level Multiplayer API uses this object but MultiplayerPeer can be extended to handle specific networking needs

CONSIDER THESE SCENARIOS…

- A player performs a jump
- A player is hit by ground spikes
- A player attacks another player

1. Where is action initiated?
2. How is action processed?
3. What view of the action do the server/clients need?

USEFUL RPC DOCUMENTATION

- RPC specification for Nodes here: https://docs.godotengine.org/en/4.1/classes/class_node.html
- Multiplayer API documentation here: https://docs.godotengine.org/en/stable/classes/class_multiplayerapi.html
- Multiplayer Peer documentation here: https://docs.godotengine.org/en/stable/classes/class_multiplayerpeer.html

RESOURCES

- <http://mauve.mizuumi.net/2012/07/05/understanding-fighting-game-networking.html>
- <https://docs.godotengine.org/en/3.1/tutorials/networking/high_level_multiplayer.html>
- <https://github.com/devonh/Godot-engine-tutorial-demos/tree/master/2018/07-30-2018-multiplayer-high-level-api>
- <https://mrminimal.gitlab.io/2018/07/26/godot-dedicated-server-tutorial.html>