Networking

Matěj Vrba

Content

- How to start with networking
- Core concepts
- Keywords
- Unreal Objects in multiplayer
- RPC
- Mindset
- Latency
- Lag compensation
- Cheating

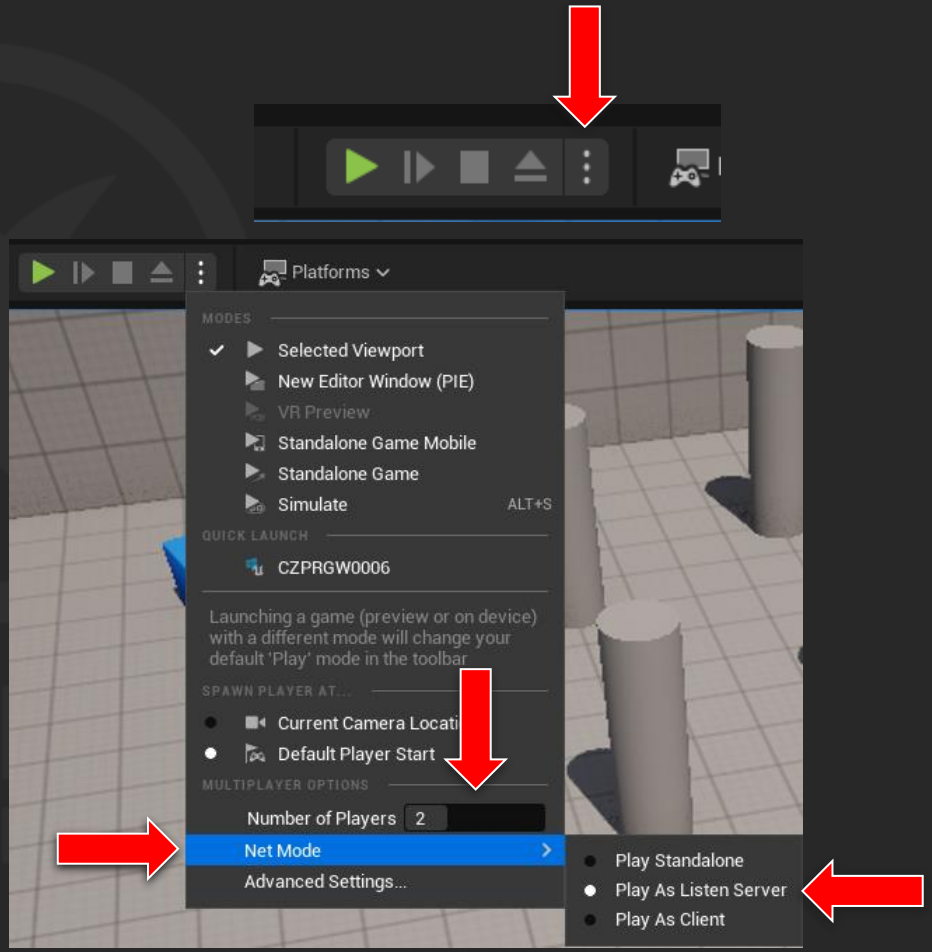
- Parts of multiplayer
  - Backend
  - Matchmaking
  - P2P Relays
  - Data Centers

- Debugging
- Optimization

- Simple LAN game vs AAA

How to start with Networking

- Multiple local clients
- Listen server mode
- What is travel?
- What’s happening in background?
  - Handshake
  - Protocol Version



Core concepts

- Net Modes:
  - Standalone
    - Client + Server Locally
  - Client
    - Client code only
  - Listen Server
    - Client + Server open to join
  - Dedicated Server
    - Server only, open to join

Core concepts

- Net Roles
  - Authority
    - I’m saying where and what I am
  - Simulated Proxy
    - I’m showing actor and simulating movement
  - Autonomous Proxy
    - Me as a player on server, I control my movement, but I don’t have authority over data
  - None
    - Not Replicated

Keywords

- Connection
  - Socket “wrapper”
- NetDriver
  - Connection manager
- RPC
  - Remote Procedure Call
- Beacon
  - Tool for communication with server without all the game code and travel around it
- Channel
- Replication
  - Keeping same value in field across network
- Desync

Replication

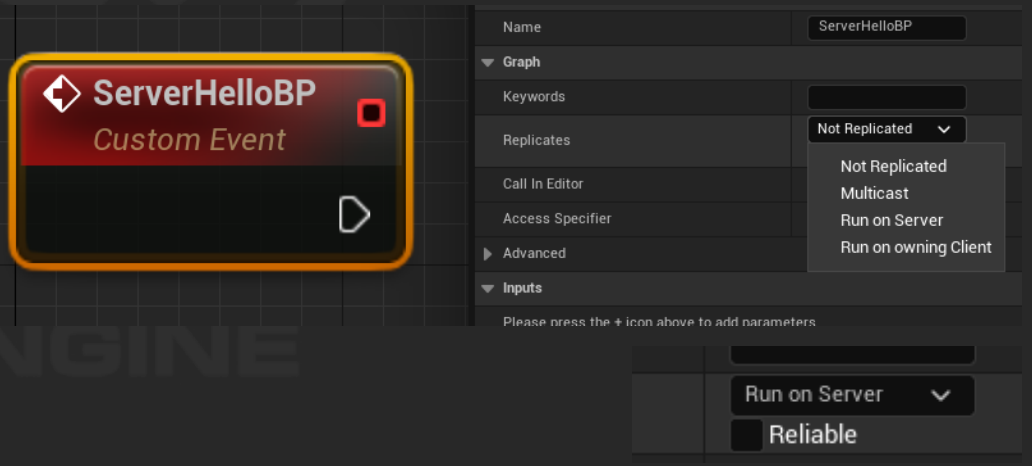
```
public:

    void GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const;

    UPROPERTY(Replicated, ReplicatedUsing=Health_OnRep)
    float Health;

    UFUNCTION()
    void Health_OnRep(float OldValue);
```

```
void APGPlayerState::GetLifetimeReplicatedProps(TArray<FLifetimeProperty>& OutLifetimeProps) const
{
    Super::GetLifetimeReplicatedProps([&]OutLifetimeProps);

    DOREPLIFETIME(APGPlayerState, Health)
}
```

RPCs

- Remote Procedure Call
- Imagine as sending direct message
- Types:
  - Client
  - Server
  - NetMulticast
- Can’t return
- Controller = Imagine as player

```
UFUNCTION(Server, Reliable)
void ServerHello(const FString& Name);


UFUNCTION(Client, Unreliable)
void ClientAnswerUnreliable(const FString& Message);
```



RPCs

```cpp
void APGMenuController::ServerHello_Implementation(const FString& Name)
{
    const FString Answer = FString::Format(TEXT("Hello {} from server"), InOrderedArguments: {Name});

    ClientAnswerUnreliable(Answer);
}

void APGMenuController::ClientAnswerUnreliable_Implementation(const FString& Message)
{
    UE_LOG(LogTemp, Log, TEXT("%s"),*Message);
}
```

RPCs

RPC invoked from the server

Actor ownership	Not replicated	`NetMulticast`	`Server`	`Client`
Client-owned actor	Runs on server	Runs on server and all clients	Runs on server	Runs on actor's owning client
Server-owned actor	Runs on server	Runs on server and all clients	Runs on server	Runs on server
Unowned actor	Runs on server	Runs on server and all clients	Runs on server	Runs on server

RPCs

RPC invoked from a client

Actor ownership	Not replicated	`NetMulticast`	`Server`	`Client`
Owned by invoking client	Runs on invoking client	Runs on invoking client	Runs on server	Runs on invoking client
Owned by a different client	Runs on invoking client	Runs on invoking client	Dropped	Runs on invoking client
Server-owned actor	Runs on invoking client	Runs on invoking client	Dropped	Runs on invoking client
Unowned actor	Runs on invoking client	Runs on invoking client	Dropped	Runs on invoking client

Unreal Objects in multiplayer

- GameState
  - Server and clients
- Gamemode
  - Server only
- PlayerState
  - Server and clients
- PlayerController
  - Server and owning client
- GameInstance
  - Server has its own, client has its own
  - Not replicated

Mindset

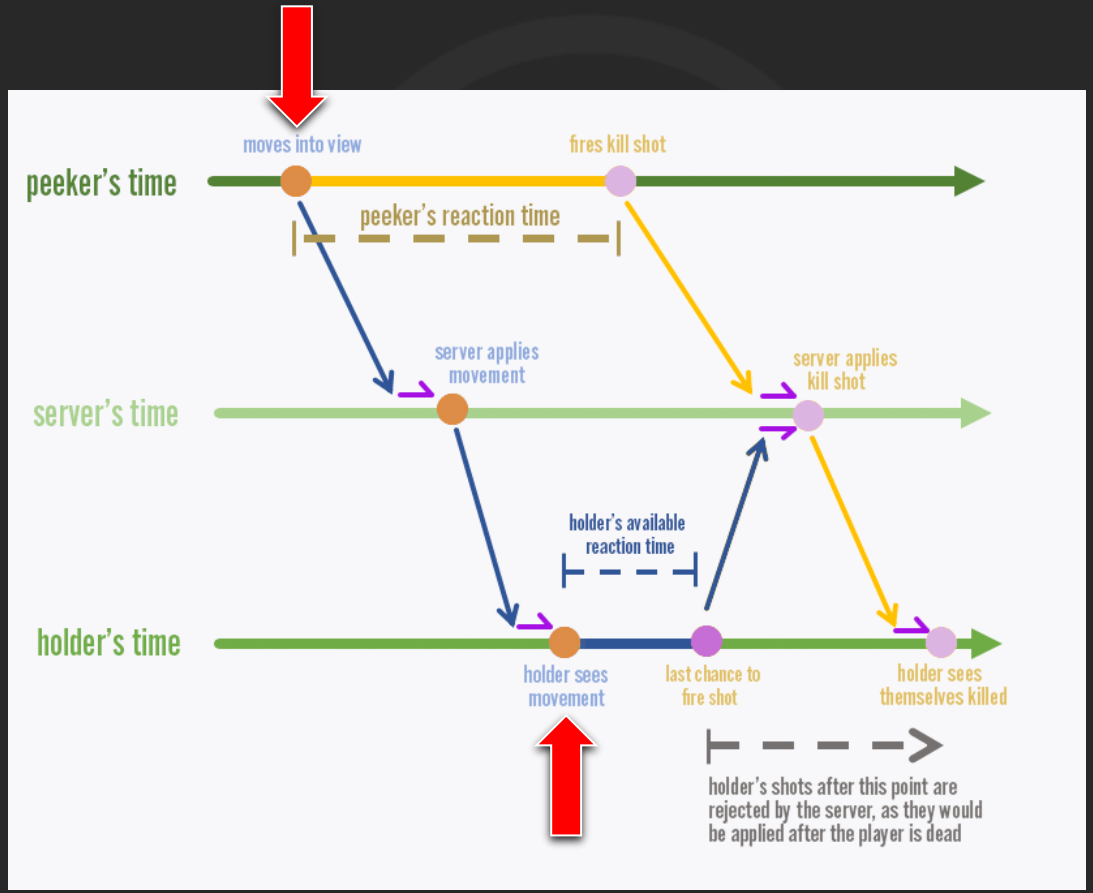
- As a client:
  - Limited data
  - I can’t do much outside of RPCs
  - When I do something, I have to tell others
- As a server:
  - I do everything (from logic side)
  - I have to tell clients about stuff
  - I have game state, I manage that
  - Network problems happen
- As a developer:
  - What if this timing issue happens
  - If I want to sabotage, what can I do?

Latency

- Main latency sources
  - Network
  - Client
  - Server
  - Arbitrary latency
- “I shot him on my monitor” problem
- You play in future, other players are in past
- Only server has real state of game
- Deathboxes

Latency

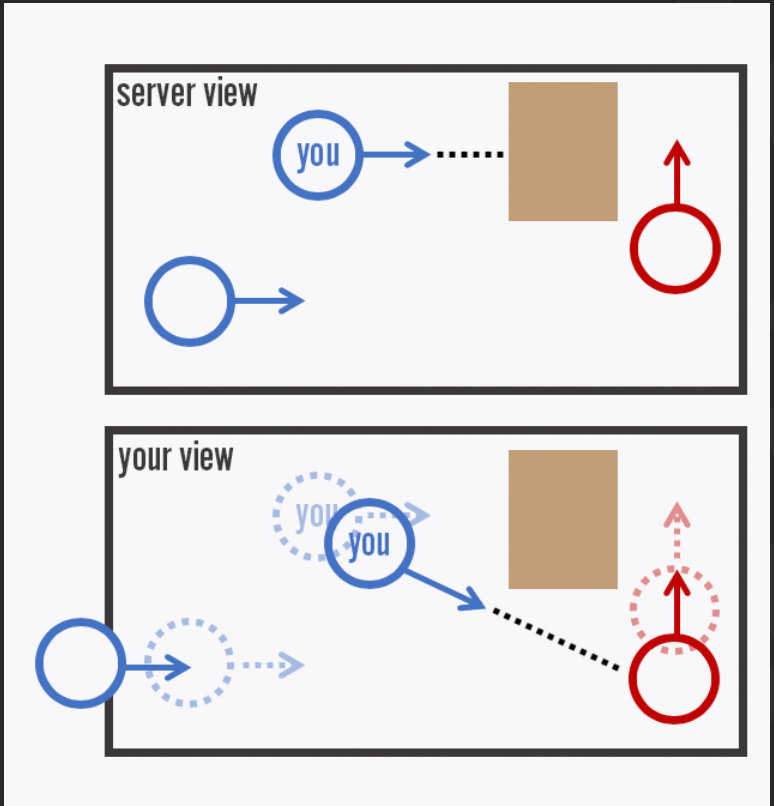
Latency


moves into view
fires kill shot
peeker's time
peeker's reaction time
server applies movement
server applies kill shot
server's time
holder's available reaction time
holder's time
holder sees movement
last chance to fire shot
holder sees themselves killed
holder's shots after this point are rejected by the server, as they would be applied after the player is dead

Lag compensation

- Multiplayer is async
- What would it look like, if there was no lag compensation?
- Rollbacks
- Arbitrary input delay
  - 1v1 fighter games
  - GDC Talk
- Predictions
- Deterministic physics
- High or Low ping players have advantage (Example)
- Rocket League networking
- Maximum supported latency

Lag compensation



Cheating

- Client side scripting
- Packet modification
  - Dropping
  - Reordering
  - Editing
- Detection
- Protection
  - Client side protection (Easy Anti-Cheat, Vanguard, BattlEye)
  - Validation
  - Replays + Human verification
  - Game design
  - Don’t give clients data that they don’t need (Fog of war)
- Banning

Cheating


M
165 S 195 210 SW 240 255 W 285 300 NW 330 345 N 15 30 NE 60
294
8 SQUADS LEFT 21
ROUND 1 - CLOSING 2:18
Theeeeeon
260M
HaSaG1_AsAyO
Helmet Lv2
[28m]
Selectfire v2
[23m]
37m 4&Lifeline
38m 43m 4&Bloodhound
76m 15&Loba
Theeeeeon
HaSaG1_AsAyO
idkukeep
4
D
Z
G
55
234
1 SPITFIRE

Parts of multiplayer

- Backend
  - Data storage
  - Trusted authority
- Matchmaking
  - Algorithm that connects players to sessions
  - Can be simple
  - Tightly connected to P2P Relays / Data centers
- P2P Relay
  - Steam and Epic can do it for you
- Data centers
  - Maximum supported latency
  - Locations

Backend

- Player data storage
  - Owned skins
  - Currency
  - Statistics
  - Match history
- Matchmaking
- UGC
- Friend list
- Ingame events
- Managing dedicated servers

Matchmaking

- Connects players in sessions
- Session != Server
- Does not require backend
- UE Online Subsystem
  - Steamworks
  - Epic Online Services
- Server browser
- Skill rating (Elo, MMR)
- Closely connected to P2P Relay or Data centers

P2P Relays and Data Centers

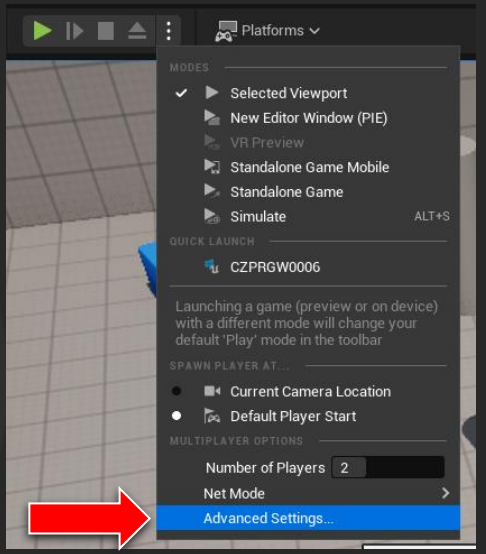
- P2P
  - NAT
  - NAT punching
  - STUN protocol
- Dedicated servers
  - Coverage
  - Cost
    - Traffic
    - CPU
    - Storage
  - Scalability
  - Trying to run as many servers as possible on single CPU

Simple LAN game vs AAA

- Backend vs No backend
- Dynamic scaling vs P2P
- Full server authority, client only sends commands to server
- Matchmaking with matching skill group

Debugging

- Unreal Net simulation
- Unreal networking profiler
- Custom tools over beacon
  - Example: Send server tick duration



Optimization

- N*N-1 Complexity
- Can be reduced by game design

- Distance culling
- Replication frequence

- Less data to replicate = better
  - Minecraft: Chunk update vs block update
  - Minecraft: Don’t send chest inventory until needed

Good networking guides

- GDC talks on youtube
- Unreal Docs
- For low level features, UE source code
- Valorant netcode blog post