Networking in UE4

Joe Graf

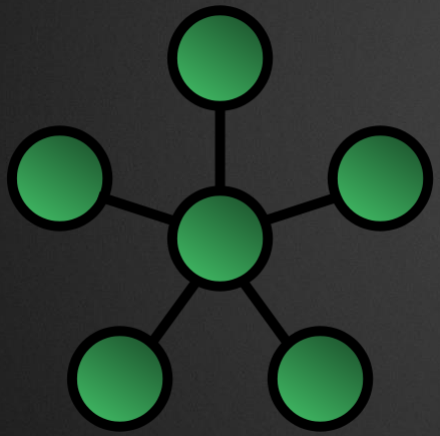
@EpicCog

Introduction

- Network topology and NATs
- C++ Replication
- Blueprint Replication
- Low Level Networking
- Diagnostic Tools

Network Topologies

Client / Server

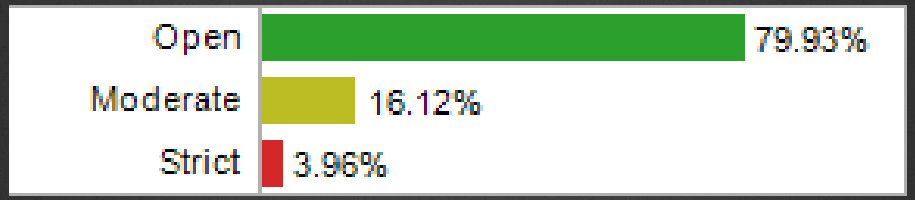


Peer to Peer

Network Address Translation (NAT)

- Three types to think about
  - Open
    - Can accept unsolicited connections
  - Moderate
    - Can sometimes accept unsolicited connections
  - Strict
    - Can only accept connections from known addresses

Gears of War 3 NAT Analytics

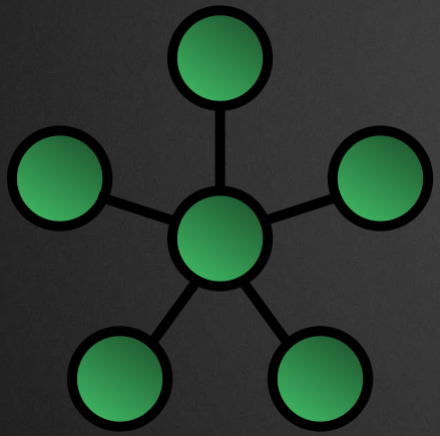


NAT Implications

- Client/Server
  - Server should have Open NAT
  - Client can be any NAT type
- Peer to Peer
  - All clients need Open NATs
  - ~20% of players can't play your game

Server Authoritative

Client / Server



- Clients only talk to Server
- Server replicates to Clients
- Data flow is Server->Clients
- RPCs are bidirectional

C++ Replication

Basic Gameplay Classes

Server

- GameMode
- GameState
- PlayerController
  - One for each player
- PlayerState
  - One for each player

Client

- GameState
- PlayerController
  - One for each **local** player
- PlayerState
  - One for each player

Gameplay Classes

- GameMode
  - Rules for your game
- GameState
  - Replicated information about your GameMode

Gameplay Classes (cont)

- PlayerController
  - Primary interaction with your GameMode
- PlayerState
  - Replicated information about your player

Data Replication

- bReplicates
  - Set as part of the Actor's construction
- UPROPERTY Macro
  - Replicated
  - ReplicatedUsing
- GetLifetimeReplicatedProps Function
  - Determines the set of properties for replication

bReplicates

```
ACoolActor::ACoolActor(…) :
      Super( PCIP )
{
      bReplicates = true;
}
```

Replicated Example

```
/** number of kills */
UPROPERTY(Transient, Replicated)
int32 NumKills;
```

ReplicatedUsing Example

```
/** team number */
UPROPERTY(Transient,
      ReplicatedUsing=OnRep_TeamColor)
int32 TeamNumber;

void AShooterPlayerState::OnRep_TeamColor()
{
      UpdateTeamColors();
}
```

GetLifetimeReplicatedProps Example

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

  DOREPLIFETIME( AShooterPlayerState, TeamNumber );
  DOREPLIFETIME( AShooterPlayerState, NumKills );
  DOREPLIFETIME( AShooterPlayerState, NumDeaths );
}
```

Conditional Replication

- Property is only replicated when a condition is met
- Can reduce bandwidth consumed
- Done via a similar macro

Conditional Replication Example

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

  DOREPLIFETIME_CONDITION(AMyPlayerState,
      SomeOwnerOnlyVal,
      COND_OwnerOnly);
}
```

Common Replication Conditions

- COND_InitialOnly
  - This property will only attempt to send on the initial replication
- COND_OwnerOnly
  - This property will only send to the actor's owner
- COND_SkipOwner
  - This property send to every connection EXCEPT the owner
- COND_SimulatedOnly
  - This property will only send to simulated actors
- COND_AutonomousOnly
  - This property will only send to autonomous actors

Function Replication

- UFUNCTION Macro
  - Reliable
  - Unreliable
  - Client
  - Server
  - NetMulticast
  - WithValidation
  - BlueprintAuthorityOnly
  - BlueprintCosmetic

Reliable

- Function is guaranteed to be called
- Resent when an error is present
- Delayed when bandwidth is saturated

Reliable Example

```
/** notify player about started match */
UFUNCTION(Reliable, Client)
void ClientGameStarted();

void AShooterPlayerController::ClientGameStarted_Implementation()
{
  bAllowGameActions = true;

  // Enable controls and disable cinematic mode now the game has started
  SetCinematicMode(false, false, true, true, true);

  AShooterHUD* ShooterHUD = GetShooterHUD();
  if (ShooterHUD)
  {
    ShooterHUD->SetMatchState(EShooterMatchState::Playing);
    ShooterHUD->ShowScoreboard(false);
  }
  …
}
```

Unreliable

- Function is attempted to be sent
- Not sent again when an error is present
- Skipped when bandwidth is saturated

Unreliable Example

```
/** Replicated function sent by client to server - contains
client movement and view info. */
UFUNCTION(Unreliable, Server, WithValidation)
virtual void ServerMove(float TimeStamp, …);

void UCharacterMovementComponent::ServerMove_Implementation(
        float TimeStamp, …)
{
        …
}
```

NetMulticast

- Sends to all clients
  - Reliable or Unreliable applies here too

NetMulticast Example

```
/** broadcast death to local clients */
UFUNCTION(Reliable, NetMulticast)
void BroadcastDeath(…);

void AShooterPlayerState::BroadcastDeath_Implementation(…)
{
  for (auto It = GetWorld()->GetPlayerControllerIterator(); It; ++It)
  {
    // all local players get death messages so they can update their huds.
    AShooterPlayerController* TestPC = Cast<AShooterPlayerController>(*It);
    if (TestPC && TestPC->IsLocalController())
    {
      TestPC->OnDeathMessage(KillerPlayerState, this, KillerDamageType);
    }
  }
}
```

WithValidation

- Called before the target function
- Used to validate parameters of a function
  - Meant to detect cheating/hacking
- Return value affects whether function is called
  - `false` skips the call and kills the connection

WithValidation Example

```
bool UCharacterMovementComponent::ServerMove_Validate(
        float TimeStamp, …)
{
  bool bIsValidMove = false;

  // Perform move validation here

  return bIsValidMove;
}
```

BlueprintAuthorityOnly

- UE4 all are functions are Simulated
  - This is the opposite of UE3

- Opt out via BlueprintAuthorityOnly

BlueprintAuthorityOnly Example

```
UFUNCTION(BlueprintImplementableEvent,
    BlueprintAuthorityOnly,
    meta=(FriendlyName = "AnyDamage"),
    Category="Damage")
virtual void ReceiveAnyDamage(float Damage,
    const UDamageType* DamageType,
    AController* InstigatedBy,
    AActor* DamageCauser);
```

BlueprintCosmetic

- Opposite of BlueprintAuthorityOnly
  - Runs on the client not server

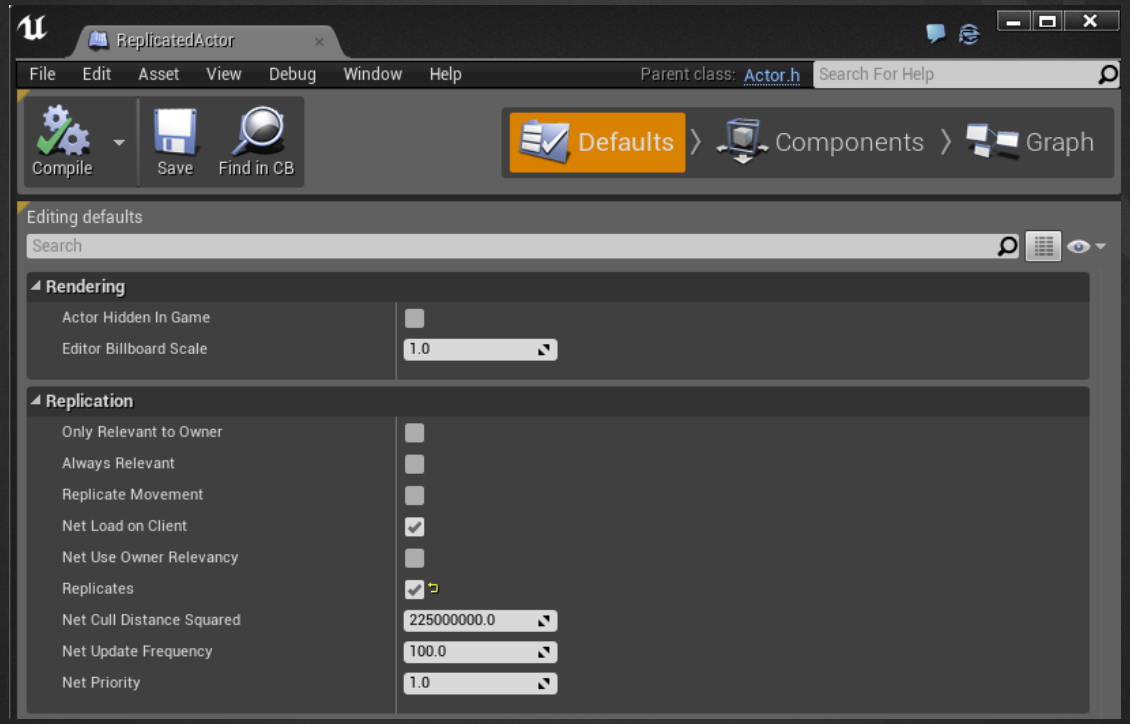
- An optimization to skip execution of slow cosmetic code on the server

Actor Relevancy

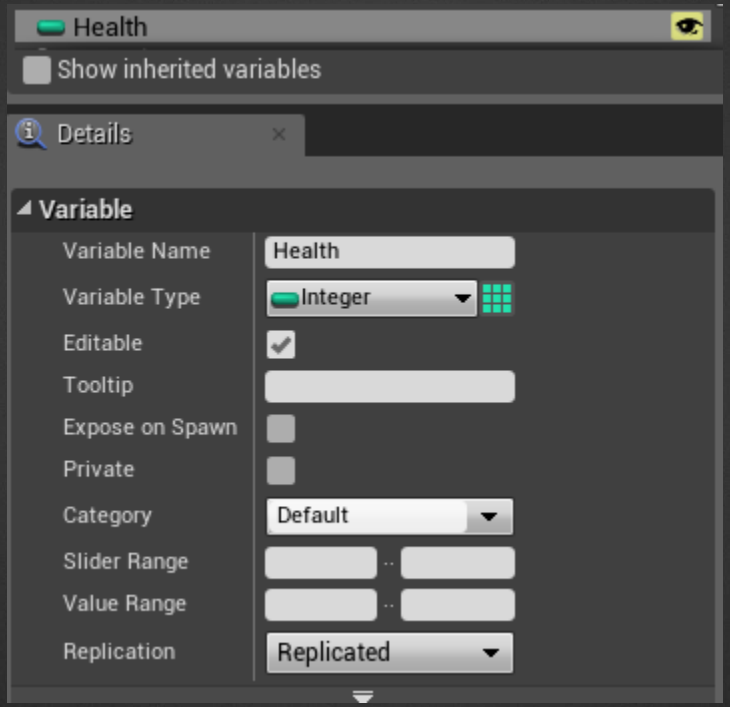
- Trades CPU time for network bandwidth
- Distance based
  - Are these actors close enough
  - Default implementation right now
- Line of sight
  - Can these actors see each other
  - UE3 default implementation
- Always relevant is an option
  - Trades bandwidth for CPU time

Blueprint Replication

Actor Replication

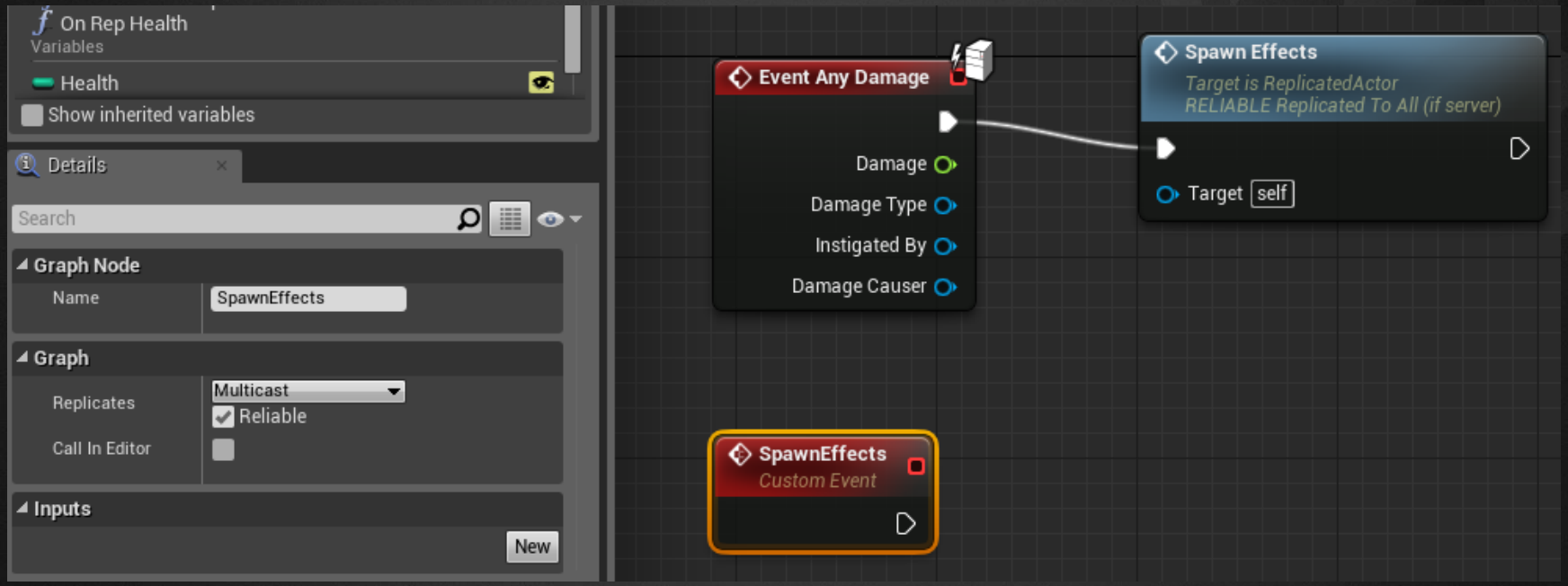


Property Replication



Property Replication Notify

Function Replication



Low Level Details

Low Level Implementation

- UNetDriver
- UNetConnection
- UChannel
  - UControlChannel
  - UVoiceChannel
  - UActorChannel

UNetDriver

- Contains a list of connections to Tick
- On client, one connection
- On Server, *N* connections

UNetConnection

- Contains a list of channels to replicate

- UChildConnection
  - Used as an optimization for split-screen games

UChannel Objects

- Logical construct
  - Routes data to the proper object
- Accessed by ChannelID
  - Some channels have predefined IDs

UControlChannel

- Handles connection handshaking
- Processes object loading requests
- Deals with misc. non-gameplay communication

UVoiceChannel

- Sends and receives voice data
  - Voice channel routes data to platform handler
- Voice data is platform dependent
- Voice data is sent as speaker ID and payload

UActorChannel

- Handles actor replication
  - Includes any replicated components
- One actor channel for each replicated actor
- Actors are replicated by channel ID
  - Dynamically assigned based upon array position

Voice Considerations

- Game can choose to support or not
- Platform can mute other players
  - Player muted another outside of the game
- Players can mute other players
- Gameplay can mute players
  - Team based, distance based, push to talk

Voice Functions

```
UFUNCTION(Server, Reliable, WithValidation)
virtual void ServerMutePlayer(FUniqueNetIdRepl PlayerId);
UFUNCTION(Server, Reliable, WithValidation )
virtual void ServerUnmutePlayer(FUniqueNetIdRepl PlayerId);
UFUNCTION(Reliable, Client)
virtual void ClientMutePlayer(FUniqueNetIdRepl PlayerId);
UFUNCTION(Reliable, Client)
virtual void ClientUnmutePlayer(FUniqueNetIdRepl PlayerId);

/**
 * Mutes a remote player on the server and then tells the client to mute
 *
 * @param PlayerNetId the remote player to mute
 */
void GameplayMutePlayer(const FUniqueNetIdRepl& PlayerNetId);
/**
 * Unmutes a remote player on the server and then tells the client to unmute
 *
 * @param PlayerNetId the remote player to unmute
 */
void GameplayUnmutePlayer(const FUniqueNetIdRepl& PlayerNetId);
```

Diagnostic Tools

Network Logging

- LogNet
  - Verbose information about the state of channels and connections
- LogNetPlayerMovement
  - Detailed information about movement from clients and corrections from server
- LogNetTraffic
  - Verbose information about data sent on a connection

Network Statistics

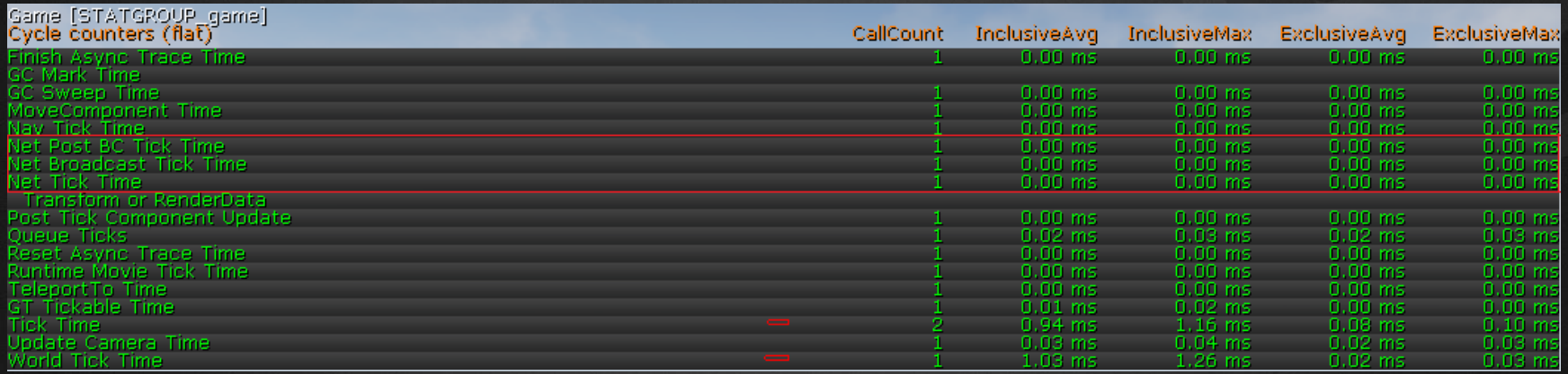
- Stat Net
  - Lists ping, channel count, in/out bytes, etc.
- Stat Game
  - List of network processing information

Stat Net Example

Net [STATGROUP_NET]

Counters	Average	Max
Channels		98.00
In Bunches		61.00
In Loss		0.00
In Packets		30.00
In Rate (bytes)		2624.00
NetGUID In Rate (bytes)		0.00
NetGUID Out Rate (bytes)		0.00
Saturated		1.00
Num Actor Channels		95.00
Num Chan ready for dormancy		55.00
Num Actors		1284.00
Num Considered Actors		94.00
Num Dormant Actors		0.00
Num Initially Dormant Actors		0.00
Num Network Actors		98.00
Num ACKd NetGUIDs		164.00
Num Pending NetGUIDs		0.00
Num UnACKd NetGUIDs		0.00
Num Relevant Actors		38.00
Num Relevant Deleted Actors		0.00
Num Replicated Actor Attempts		38.00
Num Replicated Actors Sent		13.00
Out Bunches		355.00
Out Loss		0.00
Out Packets		34.00
Out Rate (bytes)		9983.00
In % Voice		0.00
Out % Voice		0.00
Ping		0.00
Num Prioritized Actors		95.00
Voice bytes recv		0.00
Voice bytes sent		0.00
Voice packets recv		0.00
Voice packets sent		0.00

Stat Game Example



Game [STATGROUP_game]
Cycle counters (flat)

	CallCount	InclusiveAvg	InclusiveMax	ExclusiveAvg	ExclusiveMax
Finish Async Trace Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
GC Mark Time					
GC Sweep Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
MoveComponent Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Nav Tick Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Net Post BC Tick Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Net Broadcast Tick Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Net Tick Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Transform or RenderData					
Post Tick Component Update	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Queue Ticks	1	0.02 ms	0.03 ms	0.02 ms	0.03 ms
Reset Async Trace Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
Runtime Movie Tick Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
TeleportTo Time	1	0.00 ms	0.00 ms	0.00 ms	0.00 ms
GT Tickable Time	1	0.01 ms	0.02 ms	0.00 ms	0.00 ms
Tick Time	2	0.94 ms	1.16 ms	0.08 ms	0.10 ms
Update Camera Time	1	0.03 ms	0.04 ms	0.02 ms	0.03 ms
World Tick Time	1	1.03 ms	1.26 ms	0.02 ms	0.03 ms

Network Simulation Options

- PktLag
  - Delays the sending of a packet by a *N*ms
- PktLagVariance
  - Provides some randomness to the PktLag option
- PktLoss
  - A percentage chance of not sending a packet

Network Simulation Options (cont)

- PktDup
  - A percentage chance of sending duplicate packets
- PktOrder
  - Sends packets out of order when enabled

Setting the Simulation Options

- Console

```
Net PktLag=100
```

- INI File

```
[PacketSimulationSettings]
PktLag=50
PktLagVariance=10
PktLoss=3
PktOrder=0
PktDup=10
```

Questions?

Documentation, Tutorials, and Help at:

- AnswerHub: http://answers.unrealengine.com
- Engine Documentation: http://docs.unrealengine.com
- Official Forums: http://forums.unrealengine.com
- Community Wiki: http://wiki.unrealengine.com
- YouTube Videos: http://www.youtube.com/user/UnrealDevelopmentKit
- Community IRC: #unrealengine on FreeNode

Unreal Engine 4 Roadmap

- lmgtfy.com/?q=Unreal+engine+Trello+