Proyecto
Lab21

Grupo
Dire Wolf Games

DISEÑO TÉCNICO DE FUNCIONAMIENTO DEL MOTOR DE RED

Hito:
Fecha entrega: 12-01-2017
Versión: 2.0

Componentes:
- Aarón Colston Avellà Hiles
- Sergio Huertas Ferrández
- Eduardo Ibáñez Frutos
- Marina López Menárguez
- Rubén Moreno Mora
- Rafael Soler Follana

Contenido

1. Introducción

1.1. Propósito

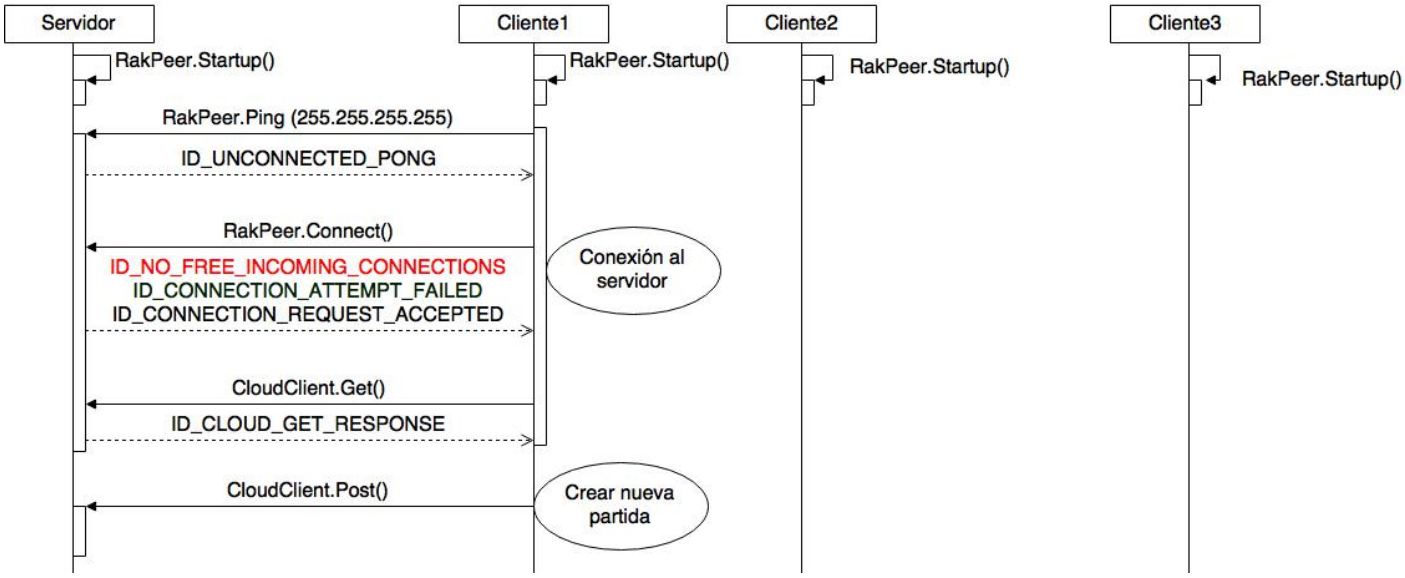
Desarrollar un sistema de control para multijugador en red para aplicarlo en el Videojuego. Hemos elegido el motor de red RakNet para implementar este sistema, el cual posee licencia de uso BSD que permite su uso, modificación y distribución libre manteniendo el copyright de la empresa Oculus VR.

En este documento se pretende detallar los esquemas y diagramas de implementación del motor y la comunicación entre los distintos nodos de la red.

2. Diagrama de Secuencia de Conexión

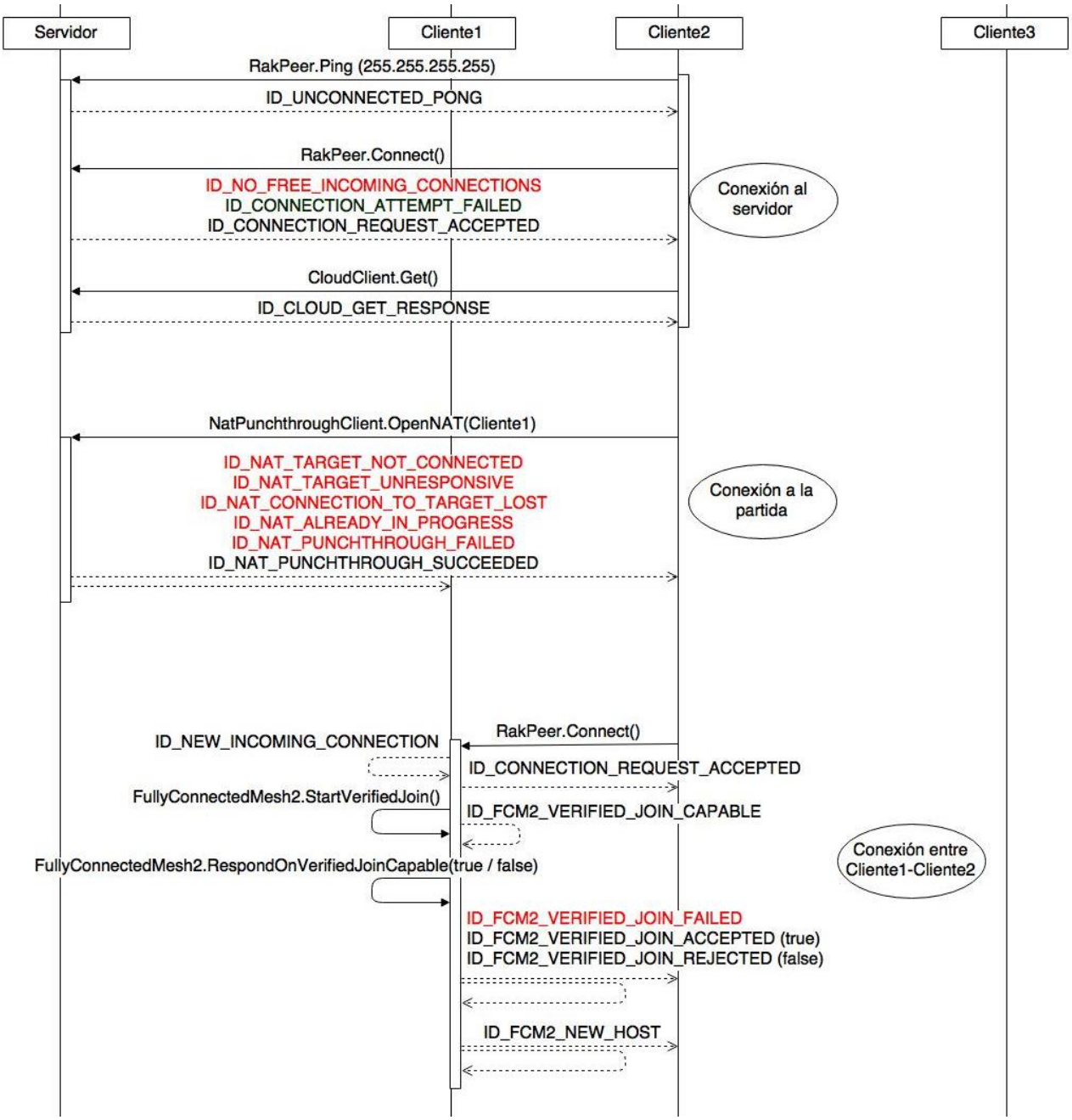
2.1. Conexión del primer cliente.

El cliente lanza un broadcast de un ping, y recibe un mensaje ID_UNCONNECTED_PONG de cada uno de los servidores que encuentre en la red. Una vez que se tienen las direcciones de los servidores se realiza la petición de conexión a uno de ellos, recibiendo la respuesta pertinente. A continuación, el cliente pregunta por las demás instancias gestionadas por CloudServer. Por último, el cliente envía al CloudServer un mensaje de creación de nueva partida.



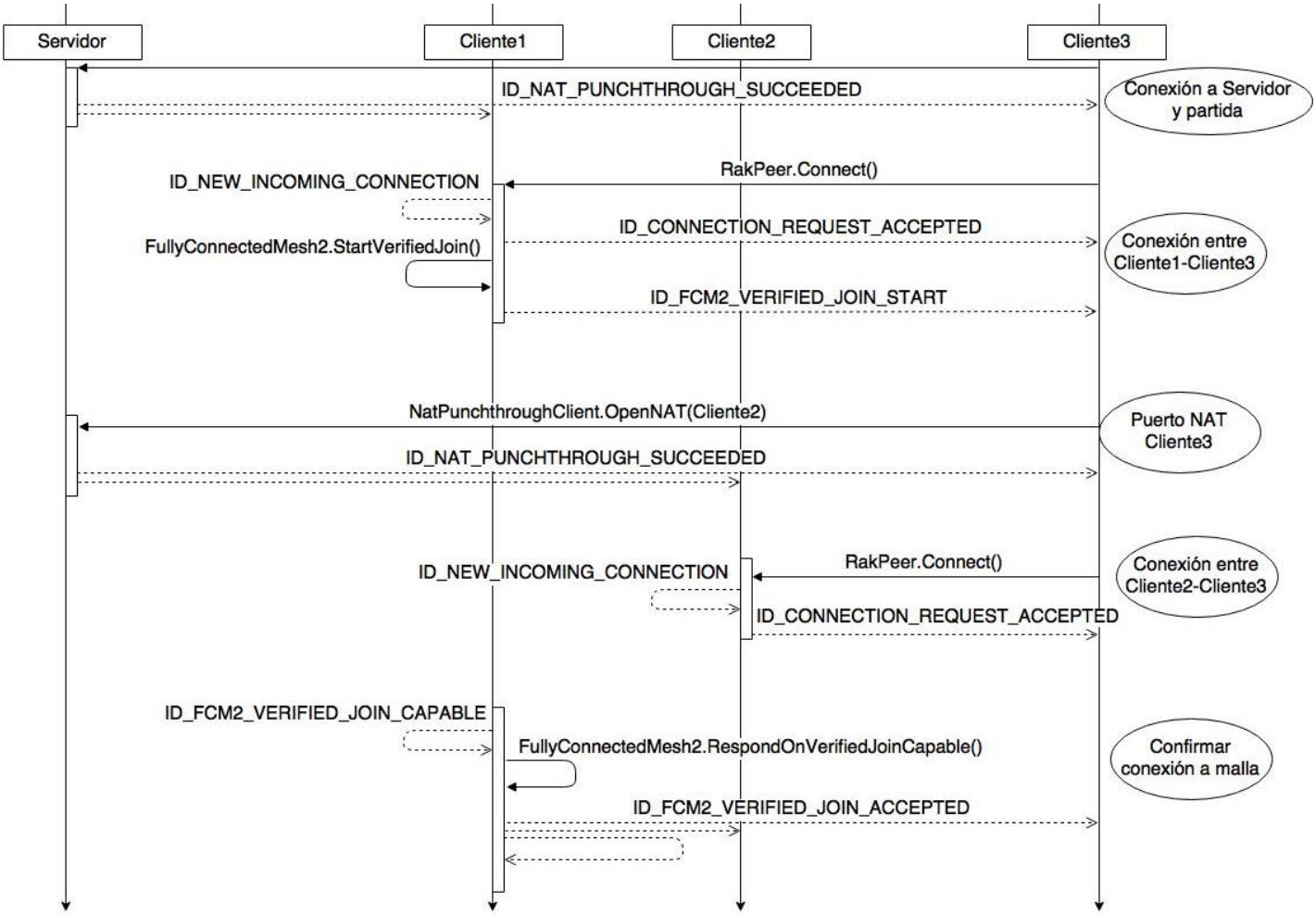
2.2. Conexión de un segundo cliente.

El cliente realiza los mismos pasos para la conexión al servidor descritos en el punto anterior. Una vez conectado con el servidor, se conecta con la partida. Primero se hace una petición al NatPunchthroughServer de OpenNAT, que configura la tabla de enrutamiento para la conexión (sin conectar) con el cliente especificado, en este caso el Cliente1. A continuación se realiza la conexión entre el primer y el segundo cliente.



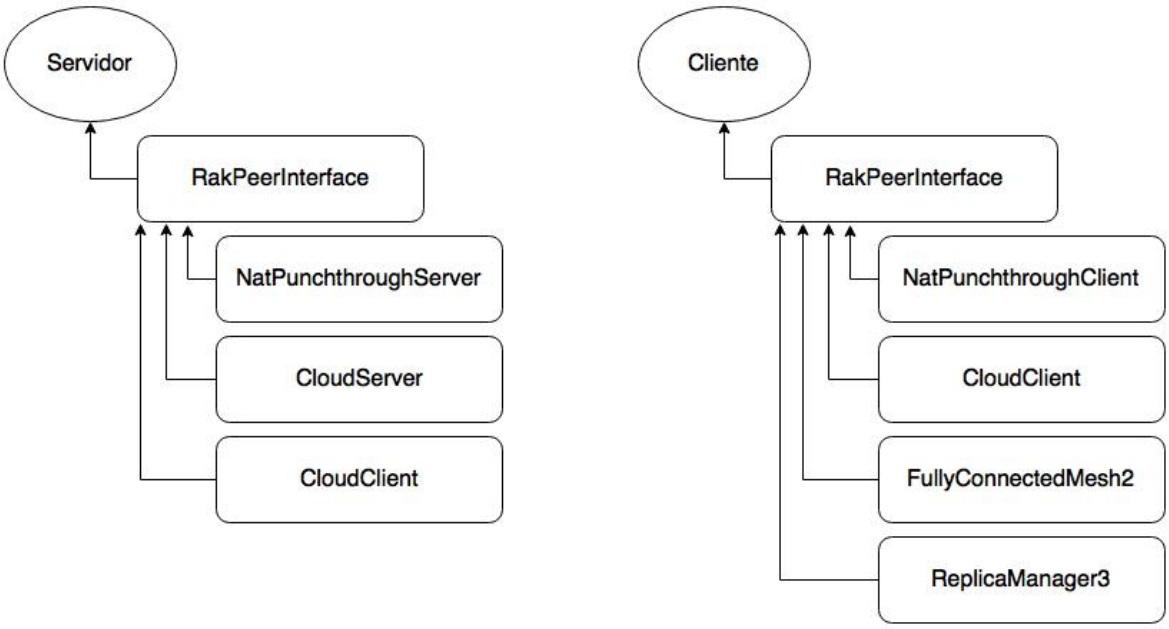
2.3. Conexión de un tercer cliente.

El cliente realiza los mismos pasos para la conexión al servidor y a la partida descritos en el punto anterior, esta vez no indicado en el diagrama, describiendo a partir de la conexión entre el Cliente3 y el Cliente1 (conexión a la partida). A continuación, el Cliente3 lanza una petición para configurar la tabla de enrutamiento entre el Cliente3 y el Cliente2 y se conecta a este, recibiendo la confirmación por parte de Cliente1 de la conexión a la malla de nodos que conforma la partida.



3. Estructura de objetos.

3.1. Objetos servidor / cliente.

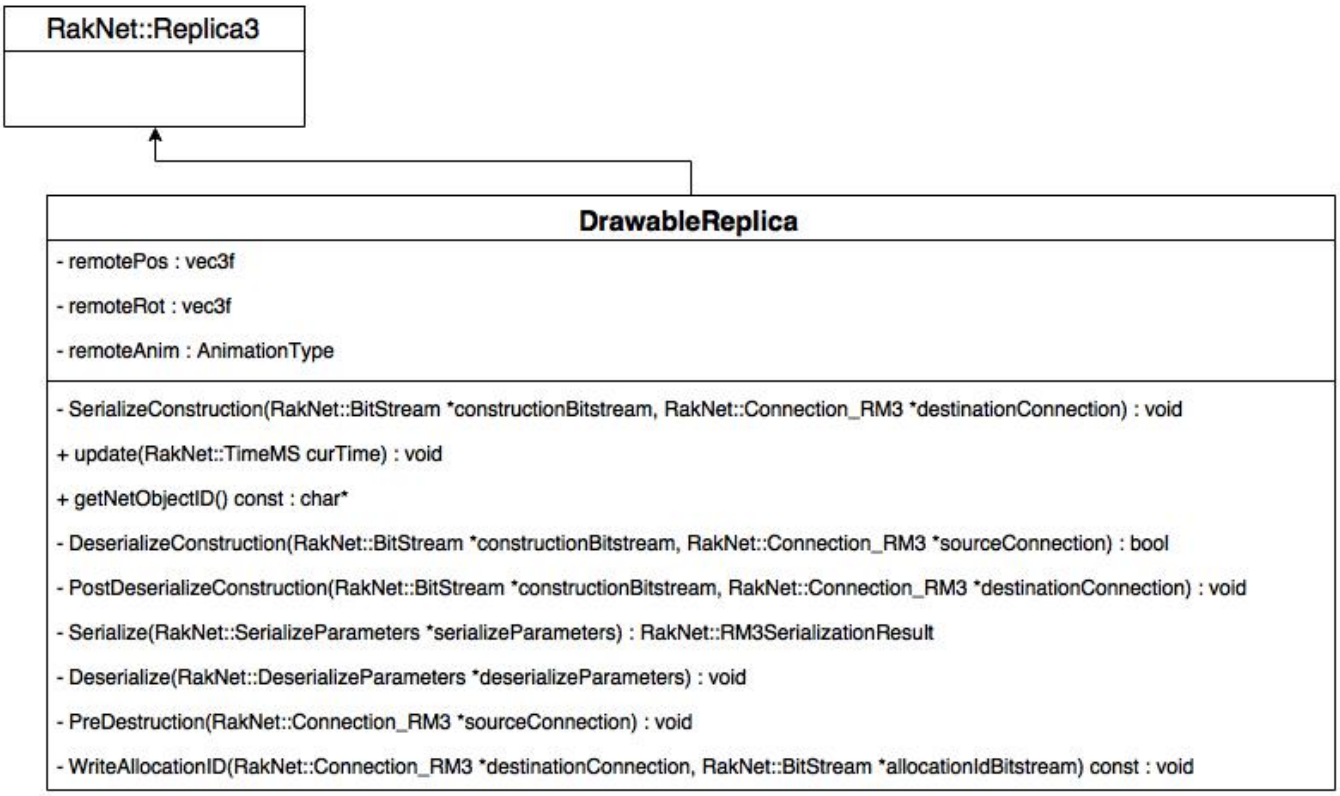


RakPeerInterface: Interfaz principal de RakNet para gestionar las comunicaciones.

Plugins de RakPeerInterface utilizados:

- *NatPunchthroughServer: Configura la conexión entre máquinas que están detrás de Routers.*
- *NatPunchthroughClient: Permite la comunicación con NatPunchthroughServer para configurar el mapa de enrutamiento entre máquinas.*
- *CloudServer: Almacena los datos de cada uno de los clientes, permitiendo compartirla entre varios servidores.*
- *CloudClient: Para la comunicación entre los distintos CloudClient de las máquinas.*
- *FullyConnectedMesh2: Conecta RakPeer con todas las demás conexiones y determina que cliente es el host (el que lleva más tiempo conectado).*
- *ReplicaManager3: Gestiona la creación, actualización y destrucción de los objetos replicables en red (los heredados de Replica3)*

3.2. Replicación de objetos.



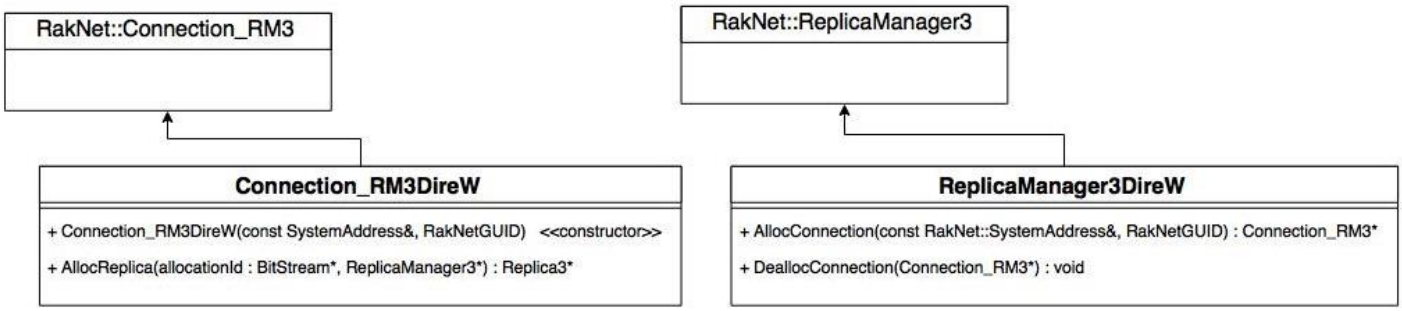
- *getNetObjectID: devuelve el nombre del objeto remoto para su identificación.*
- *update: actualiza las propiedades locales con los valores remotos.*
- *SerializeConstruction: parámetros de construcción que se le pasan a las demás instancias del juego al crear el objeto.*
- *DeserializeConstruction: parámetros de construcción que son recibidos desde las demás instancias.*
- *PostDeserialize: se ejecuta después del DeserializeConstruction. Se inicializan los valores locales desde los remotos recibidos en DeserializeConstruction.*
- *Serialize: se escribe en un bitstream los valores a replicar, se envían a las demás instancias si han cambiado.*
- *Deserialize: se reciben los valores enviados en los Serialize de las demás instancias.*
- *PreDestruction: cuando se va a destruir un objeto remoto.*
- *WriteAllocationID: escribimos el nombre del objeto remoto.*

3.3. Clase principal.

NetGame

+ isConnectedToNATPunchthroughServer : bool <<static>>
- _max_players : const int <<static>>
- _tcp_port : const unsigned short <<static>>
- _udp_sleep_timer : const RakNet::TimeMS <<static>>
- _time_search_server : const unsigned int <<static>>
- multiplayer : bool
- connected : bool
- connectionFailed : bool
- connectionRejected : bool
- gamesSearched : bool
- isServer : bool
- gameStarted : bool
- participantOrder : unsigned short
- serverIP : string
- scene : Scene*
- cloudServerGUID : RakNetGUID
- netEntities[] : Entity*
- numNetEntities : unsigned int
- netConsumables[] : Consumable*
- numNetConsumables : unsigned int
- netEnemies[] : Enemy*
- numNetEnemies : unsigned int
- netEnemyIndex : unsigned int
- servers : vector<string>
- gamesIP : vector<string>
- gamesGUID : vector<RakNetGUID>
- rakPeer : RakPeerInterface*
- networkIDManager : NetworkIDManager*
- replicaManager3 : ReplicaManager3DireW*
- natPunchthroughClient : NatPunchthroughClient*
- cloudClient : CloudClient*
- fullyConnectedMesh2 : FullyConnectedMesh2*

+ Instance() : NetGame* <<static>>
+ open(Scene *, multiplayer : bool) : void
+ close() : void
+ update() : void
+ addNetObject(dwn::DrawableReplica *) : void
+ addNetEntity(Entity*) : void
+ addNetConsumable(Consumable*) : void
+ addNetEnemy(Enemy*) : void
+ isLocalObject(RakNet::RakNetGUID) : bool
+ isMulpilayer() : bool
+ isServer() : bool
+ getPlayerMate(unsigned int) : PlayerMate*
+ getNumPlayerMates() : int
+ startGame() : void
+ searchForServers() : bool
+ connectToServer(index : unsgined int) : bool
+ connectToGame(index : unsigned int) : bool
+ sendBroadcast(messageID : unsigned int, usigned int) : void
+ sendBroadcast(messageID : unsigned int, RakString) : void
+ sendBroadcast(messageID : unsigned int, vec3f, float) : void
+ sendBroadcast(messageID : unsigned int, usigned int, vec3f, vec3f) : void
- getBitStreamEntityID(Packet*) : unsigned int
- getNatTargetName(Packet*) : RakString


RakNet::Connection_RM3
RakNet::ReplicaManager3
Connection_RM3DireW
+ Connection_RM3DireW(const SystemAddress&, RakNetGUID) <<constructor>>
+ AllocReplica(allocationId : BitStream*, ReplicaManager3*) : Replica3*
ReplicaManager3DireW
+ AllocConnection(const RakNet::SystemAddress&, RakNetGUID) : Connection_RM3*
+ DeallocConnection(Connection_RM3*) : void