

**Diseño técnico del funcionamiento del motor de red Last Bear Standing**

*Miguel Paniagua Muela*

*Miguel Córdoba Alonso*

*José María Ortiz García*

*José Roberto Martínez Gras*

*Jorge Puerto Esteban*

*Manuel Gómez Cámara*

Índice

[**Diseño de la Arquitectura**](#_w1ucr2jcgo9m)[**2**](#_w1ucr2jcgo9m)

**Diseño y lectura de paquetes** [**3**](#_qcd9gixco72u)

**Diseño técnico** [**4**](#_8byy28j0a6kt)

[Servidor](#_e0lasx5wtfwz) [4](#_e0lasx5wtfwz)

[Enviar](#_4l7kb984lb9h) [4](#_4l7kb984lb9h)

[Recibir](#_87pxq63v76k3) [4](#_87pxq63v76k3)

[Cliente](#_vbnhitwglqdw) [4](#_vbnhitwglqdw)

[Enviar](#_cjpbym3g7a2g) [4](#_cjpbym3g7a2g)

[Recibir](#_z9giztdt6sr1) [4](#_z9giztdt6sr1)

# Diseño de la Arquitectura

La arquitectura de red sigue un modelo cliente/servidor. Tanto el cliente como el servidor actúan independientemente del otro y se comunican con mensajes. Es una arquitectura diseñada para el funcionamiento en LAN. Con un máximo de 4 jugadores por partida.

La red multijugador emplea Raknet, un motor de red en C++, para facilitar la comunicación entre el cliente y el servidor, ya que Raknet proporciona los protocolos UDP y TCP.

# Diseño y lectura de paquetes

El **paquete** de información está compuesto por un **sistema de dirección**, que permite obtener información imprescindible del paquete como IP y puerto, un **GUID** que lo identifica, **tamaño** del paquete en bits y bytes y por último los **datos enviados** como char.

Como se explica en el documento de Diseño de requerimientos y funciones de red, el puerto lo asigna el cliente y el tamaño es el adecuado al mensaje por lo tanto, vamos a centrarnos en este ultimo.

# Funciones

### Servidor

Podemos dividir la función del servidor en 3 grandes bloques:

1. **CONEXIÓN AL SERVIDOR**

El servidor debe conocer siempre la cantidad de **jugadores en red**. Tras una conexión de un jugador al servidor, el servidor almacena la **IP** y el **puerto** del jugador correspondiente, se encarga de actualizar el número de jugadores en red, y envía a los clientes la ID del jugador entrante, de esta manera el jugador entrante obtiene una ID y el resto de jugadores advierten de un nuevo jugador y su ID.

1. **INICIO DE PARTIDA**

Una vez conectados los jugadores al servidor, el servidor se encarga de advertir del inicio de partida para los jugadores conectados. Para ello el servidor envía a los jugadores el mensaje correspondiente acompañado del **número de mapas** y su **orden aleatorio establecido**.

1. **ACTUALIZACION EN BROADCAST**

El servidor se encarga de flujo de mensajes de cada uno de los clientes o jugadores conectados. El servidor pues, recibe el mensaje de un determinado cliente, y en función del **puerto** e **IP** transmite este mensaje al resto de clientes conectados, permitiendo así la actualización en tiempo real de todos los clientes.

### 

### Cliente

En este caso podemos dividir la función del cliente en:

1. **CONEXIÓN AL SERVIDOR**

El cliente solicita una **IP** de conexión que almacena, y genera un puerto automáticamente de cuatro dígitos aleatorios. Una vez establecida la conexión recibe un mensaje de servidor y obtiene una ID de jugador, con la que podrá ser visto por el resto de jugadores, y las ID de los jugadores ya conectados, de esta manera obtiene el **número de jugadores en red** y sus **ID.**

1. **INICIO DE PARTIDA**

Una vez recibido el inicio de partida por parte de servidor, cliente actualiza al mundo (el propio juego), del **número de jugadores** en red, sus **ID**, **número de mapas** y **orden de aparición en la partida** y da paso al juego.

1. **ACTUALIZACION EN BROADCAST**

La manera en la que trabaja el cliente es por eventos y el juego lleva a cabo una simulación de estos eventos.

El cliente actualiza al resto de jugadores por eventos, esto es, el cliente solo envía el mensaje pertinente a través de un evento concreto. Por ejemplo, si un jugador decide saltar, cliente captará ese evento y enviará a servidor el mensaje correspondiente para notificar que el jugador saltó, el resto de clientes tras conocer esa información llevan a cabo una simulación *in game* del salto de ese jugador.

Esta forma de proceder puede generar pequeñas diferencias en los distintos clientes, por ello, cliente conoce todas las **variables de estado** de un jugador lo cual permite actualizar todo su estado cada 1-5 segundo/s al resto de clientes.