Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

AshNet: Complemento para Unity que facilita el desarrollo de videojuegos en línea

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Andrés Leone Gámez

Tutor: Ramón Pascual Mollá Vayá

Segundo tutor: Francisco José Abad Cerdá

2017-2018

Resumen

Este TFG aborda el desarrollo y despliegue de un complemento para Unity extensible y optimizado para usar pocos recursos del sistema que permite desarrollar con facilidad videojuegos en línea compatibles con todas las plataformas soportadas por Unity. Ningún complemento similar existente soporta completamente todas las plataformas de Unity, creando una oportunidad de negocio que ha llevado a desarrollar el TFG bajo el contexto de un TFG de emprendimiento a través de la plataforma Start.inf de la UPV.

Palabras clave: Unity, Online, WebRTC, Complemento, C#, Videojuego.

Abstract

This TFG deals with the development and deployment of a Unity extension extensible and optimized to use minimal system resources that allows to easily develop an online videogame compatible with all the platforms supported by Unity. No other similar extension fully supports all the platforms supported by Unity, creating a business opportunity that allowed to develop this TFG in the context of an entrepreneurship TFG through the UPV's Start.inf platform.

Keywords: Unity, Online, WebRTC, Plug-in, C#, Videogame.

Tabla de contenidos

Índice de figuras

Índice de tablas

1. Introducción

1.1. Motivación personal

Desde hace años estoy interesado en la realización de un videojuego en línea. Durante este tiempo he investigado cómo se realizan, usado complementos en el motor de videojuegos que uso, llamado Unity, e implementado código sobre dichos complementos. Para poder realizar lo anterior ha sido necesario buscar mucha información específica sobre desarrollo de videojuegos en línea, información que he encontrado en su mayoría en un blog llamado GafferOnGames[1], vídeos de conferencias de GDC[2] (*Game Developers Conference*), artículos publicados en Gamasutra[3] y documentación del propio motor de videojuegos Unity.

Tras encontrar problemas con las soluciones ya implementadas para Unity hice uso de las librerías Lidgren[4] y LiteNetLib[5], cuyo objetivo es implementar el código necesario para conectar jugadores y enviar información entre ellos, dejando al desarrollador el trabajo de implementar el protocolo de comunicación para sincronizar el estado del videojuego.

Esta solución funcionó bien, exceptuando pequeños problemas, hasta que, en junio de 2015[6], Unity añadió una nueva plataforma, llamada WebGL, que exporta el videojuego a un navegador web. Esta plataforma tenía como objetivo hacer obsoleta la plataforma WebPlayer, que usaba un complemento NPAPI para funcionar en el navegador web (como Java y Flash), con una nueva plataforma que usa únicamente tecnologías web. Este cambio fue necesario (aunque considero desafortunado, ya que un videojuego requiere de un gran rendimiento y hasta hoy, pasados 3 años, WebGL todavía no ha conseguido alcanzar el rendimiento que WebPlayer ofrecía. Por ejemplo, WebGL no permite usar múltiples hilos) porque el navegador web Google Chrome implementó una hoja de ruta para hacer obsoletos los complementos NPAPI a partir de junio de 2014[7], decisión seguida por Mozilla Firefox en junio de 2016[8]. Esta plataforma es la única incompatible con la API `Socket` ofrecida en C# y compatible con Unity y, por tanto, las dos anteriores librerías mencionadas no funcionan en esta plataforma.

La solución de Unity fue usar el protocolo WebSocket en esta plataforma en lugar de WebRTC, añadiendo problemas que se detalla más adelante. Con el tiempo aparecieron complementos de terceros con soluciones alternativas a la ofrecida por Unity, que se dividen en 3 categorías:

- Hacen uso de WebRTC, pero sólo funcionan en contadas plataformas a parte de WebGL.
- Hacen uso de WebSocket, al igual que Unity.
- No soportan la plataforma WebGL.

[1] https://gafferongames.com/
[2] http://www.gdconf.com/
[3] https://www.gamasutra.com
[4] https://github.com/lidgren/lidgren-network-gen3
[5] https://github.com/RevenantX/LiteNetLib
[6] https://forum.unity.com/threads/webgl-roadmap.334408/
[7] https://blog.chromium.org/2013/09/saying-goodbye-to-our-old-friend-npapi.html
[8] https://developer.mozilla.org/en-US/docs/Plugins/Roadmap

Ante esta situación y la necesidad de invertir tiempo en realizar este TFG llegué a la conclusión de que esta era la oportunidad ideal de solucionar este problema, implementando la API WebRTC en C# para permitir la conexión en línea entre todas las plataformas, transformándolo en un complemento para realizar videojuegos en línea que yo mismo pueda usar y finalmente vendiéndolo para poder financiar el desarrollo de futuros videojuegos.

1.2. Motivación profesional

En la actualidad y desde años atrás, los juegos en línea son de los más populares en el mercado. Estos videojuegos permiten a dos o más personas jugar entre ellas y permiten disfrutar de experiencias que no son posibles en otro tipo de videojuegos.

La elaboración de un videojuego es una tarea compleja que únicamente se complica jugar en línea es una de sus características. Esto es debido a que, para conseguir que la fantasía de jugar en línea y en tiempo real con otra persona se cumpla, es necesario replicar el estado del videojuego entre los jugadores participantes a través de internet, pero internet tiene limitaciones y se debe atener a ellas. Las limitaciones más importantes son la siguientes:

- **Latencia.** (Tiempo que tardan los paquetes en transmitirse desde un jugador a otro). El tiempo de latencia varía a lo largo del tiempo. El tiempo de latencia más pequeño teóricamente posible viene dado por la velocidad máxima a la que puede transferirse información (la velocidad de la luz), aunque en la actualidad la velocidad de transmisión de la información es inferior. Por ejemplo, la latencia mínima posible entre Valencia y Nueva York es de 20 ms[9] mientras que una medición real muestra 104 ms[10]. Para poner estos números en perspectiva, el tiempo medio de reacción humano a un estímulo visual simple es de 190 ms y a un estímulo sonoro simple es de 160 ms[11]. Debido a esta limitación las acciones tomadas por un jugador son observadas por otros jugadores con un retraso temporal dependiente de multitud de factores que pueden variar a lo largo del tiempo. Hoteles, pueblos con mala infraestructura de red y comunicación por satélite son ejemplos de redes con latencias grandes.
- **Tasa de bits.** (Cantidad de datos que se puede transmitir en la red por unidad de tiempo). La cantidad de bytes que caben en un paquete y la cantidad de paquetes que se puede enviar o recibir en una determinada cantidad de tiempo están limitadas por la red el hardware usado por el jugador y el software intermediario. En la actualidad pocos son los videojuegos cuyo estado puede ser enviado por completo entre jugadores en red, para la mayoría es necesario comprimir y optimizar la cantidad de datos necesarios para representar el estado del videojuego para poder ser transmitido a otros jugadores, siendo una práctica común enviar únicamente a un jugador la parte del estado del videojuego que le afecta, o más le afecta, y enviar estados menos importantes cuando es posible. Además, un uso excesivo del ancho de banda puede dar lugar a un incremento de latencia.
- **Pérdida y duplicación de paquetes**. Todo paquete enviado puede o no llegar, además de poder llegar múltiples veces, a su destino, pudiendo, en el peor de los casos, pasar varios minutos desde que fue enviado hasta que finalmente es recibido.

[9] 6.075 km entre Valencia y Nueva York. 6.075 km / 300.000 km/s = 20,25ms
[10] https://wondernetwork.com/pings
[11] https://en.wikipedia.org/wiki/Mental_chronometry#Types

Existe información y herramientas con la finalidad de ayudar en la creación de este tipo de videojuegos, desde conferencias de desarrolladores de videojuegos en línea exitosos hasta complementos para los motores de creación de videojuegos más importantes de la actualidad. A pesar de ello, las anteriores limitaciones limitan la complejidad del videojuego, tanto como el uso de otros recursos como CPU, GPU y memoria.

El complemento que se realizará en este TFG es para el motor de videojuegos Unity, que permite añadir lógica en un videojuego usando el lenguaje C# y complementos nativos (que deben de ser recompilados para cada plataforma soportada por el motor, siendo estas Windows, Mac, Linux, Android, iOS, PS4, Xbox one y WebGL entre otras). Este motor contiene implementada una herramienta llamada UNet cuyo fin es hacer sencillo realizar un videojuego en línea mediante la abstracción e implementación de tareas comunes para todos los videojuegos en línea. Debido a limitaciones y carencias de UNet han nacido otros complementos para Unity con la misma finalidad, pero distintos enfoques, ofreciendo alternativas que tienen distintas ventajas e inconvenientes entre ellas. UNet será hecho obsoleto en 2021 y sustituido por un nuevo complemento desarrollado por Unity a finales de 2018 (1).

1.3. Objetivos

Se realizará un complemento para Unity que permita realizar un videojuego en línea e incluya los componentes necesarios más comunes para lograrlo. Se intentará llevar a cabo los siguientes subobjetivos:

- Optimizar el uso de los recursos disponibles del sistema: ancho de banda, CPU y memoria, por orden de prioridad debido a la escasez de cada recurso.
- Replicar el estado del juego de forma subjetivamente fluida, manteniendo la mayor similitud posible entre original y réplica a lo largo del tiempo.
- Tolerar fallos en la red, como pérdida de paquetes o intentos de *hacking*.
- Soportar todas las plataformas actuales ofrecidas por Unity.
- Comercialización del producto final.
- Facilitar la extensibilidad del complemento.
- Facilitar la introducción o alteración de lógica relacionada con la sincronización en línea del estado del videojuego.
- Exponer información interna del complemento al desarrollador que lo use, para que sea posible explotar el complemento (por ejemplo, conocer sí y cuándo el otro jugador ha recibido cierta información).
- Soporte genérico como complemento C# para motores distintos a Unity.
- Hacer uso de este complemento en un videojuego.

1.4. Estructura de la obra

Esta memoria está estructurada en los siguientes capítulos:

- **Estado del arte**

Se describe la situación actual entorno al desarrollo de videojuegos en línea usando el motor Unity y listado de posibles complementos que se pueden usar para desarrollar un videojuegos en línea.

- **Análisis del problema**

Se comenta los problemas existentes y posibles soluciones en relación con los complementos existentes.

- **Evaluación de la idea de negocio**

Se analiza la rentabilidad de la solución que se llevará a cabo, prestando especial atención a qué y cuántos clientes estarían interesados además de establecer una diferenciación con la competencia existente.

- **Diseño de la solución**

Se establece el diseño del complemento en detalle antes de ser desarrollado.

- **Tecnologías utilizadas**

Se lista todas las tecnologías usadas para llevar a cabo el desarrollo de la solución propuesta, tanto de herramientas usadas como de tecnología implementada en el complemento.

- **Desarrollo de la solución propuesta**

Se hace una revisión de los pasos seguidos durante el desarrollo y problemas encontrados durante el mismo. Además, se muestra el desarrollo y resultado de los experimentos llevados a cabo por cada MVP.

- **Implantación**

Se detalla cómo hacer uso del complemento desarrollado, paso a paso.

- **Pruebas**

Se muestran las pruebas unitarias realizadas para este complemento.

- **Conclusiones**

Se concluye la memoria detallando problemas encontrados, errores cometidos, el cumplimiento de los objetivos iniciales, la relación entre este trabajo y asignaturas cursadas del grado y trabajos futuros.

- **Referencias**

Se listan las referencias bibliográficas usadas durante el desarrollo del trabajo.

- **Anexo**

Se encuentra el glosario de términos.

1.5. Convenciones

Las palabras extranjeras y rutas se remarcarán en cursiva. (*Ejemplo*)

El código fuente se muestra en letra Courier cursiva. (*`Ejemplo`*)

2. Estado del arte

El problema de la realización de un juego en línea no es nuevo y existe gran cantidad de soluciones y documentación al respecto. Hay gran variedad de distintas soluciones, tanto para Unity como para otros motores, debido a la necesidad de cubrir requisitos específicos para cada videojuego. No existe una solución genérica que sea óptima para el problema debido a la escasez de recursos de red y las grandes diferencias entre las necesidades que cada videojuego en línea tiene. Existen soluciones genéricas que permiten crear una gran variedad de videojuegos en línea de temáticas distintas, pero cada una de ellas tiene un consumo distinto de recursos y ofrece distintas funcionalidades (2). Teniendo esto en cuenta, una solución hecha a medida podría hacer un mejor uso de los recursos del sistema permitiendo balancear los recursos disponibles con mayor precisión que elegir una opción existente con un balance fijo de recursos que se aproxime a lo que se busca.

Unity, al ser un motor de videojuegos genérico, ofrece una solución genérica para la realización de videojuegos en línea llamada UNet. Debido al gran uso de recursos que realiza (2) aparecieron alternativas en forma de complementos, desde grandes a pequeños, más o menos completos y todos ellos genéricos (Por ejemplo, los complementos Photon PUN, Forge y DarkRift Networking entre otros, que se encuentran en la Asset Store de Unity[12]. Más adelante se describen estos complementos).

También existe gran cantidad de fuentes de información sobre el tema que incluyen desde técnicas de optimización hasta estructuras, diseños y pasos a seguir para la realización de diversos tipos de videojuegos en línea. Muchas de estas fuentes se encuentran en la red de forma gratuita y han sido referenciadas en la bibliografía, como vídeos y documentos donde se explican como otros han hecho un videojuego en línea (3) (4) o el creador de varios videojuegos en línea explicando cómo funcionan los videojuegos en línea y exponiendo multitud de técnicas que se pueden usar para crearlos (5).

A pesar de la existencia de librerías dedicadas únicamente a la conexión entre jugadores sin funcionalidades para replicar el estado del juego, librerías cuya utilidad como base para este TFG sobre el que empezar a trabajar serían ideales, no es posible usarlas porque están implementadas como una librería no modular, haciendo necesaria la reescritura de gran parte de las mismas con la finalidad de poder añadir las nuevas funcionalidades y características necesarias como WebRTC[13] o Steamworks (6).

Para el envío de paquetes por internet se suelen usar los protocolos TCP y UDP. TCP garantiza que todo paquete enviado llega a su destino y que además lo hace de forma ordenada, mientras que UDP no garantiza llegada u orden de los paquetes enviados. Las garantías del protocolo TCP las consigue añadiendo información extra en los paquetes en forma de cabeceras al paquete que se quiere enviar, gracias a las cuales es capaz de conocer qué paquetes han llegado al destinatario, en qué orden y si un destinatario no ha recibido algún paquete (7). Este

[12] https://assetstore.unity.com/categories/tools/network
[13] https://webrtc.org/

conocimiento es usado por el protocolo para que, en caso de la pérdida de un paquete, este sea reenviado y el receptor espere su reenvío.

Paquetes enviados por una aplicación en tiempo real suelen hacer obsoletos paquetes anteriores ya que contienen información más actualizada que la enviada en el pasado, siendo esta una situación ideal para el protocolo UDP. UDP reenvía paquetes perdidos, por tanto, no espera su reenvío retrasando el procesamiento de paquetes. Siendo los videojuegos multijugador en su mayoría aplicaciones en tiempo real, UDP es el protocolo más usado por todos ellos debido al control que este protocolo otorga frente a UDP.

Se listan a continuación distintas soluciones existentes, tanto complementos para Unity como librerías limitadas a la conexión e intercambio de información en línea que pueden ser usadas por Unity (la tabla 1 realiza una comparativa de este listado):

- **UNet (8)**

Solución *networking* estable que está incluida por defecto en Unity. Usa UDP y TCP. Implementa el soporte para establecer conexiones, un protocolo de comunicación con distintos tipos de calidad de servicio, RPC, *host migration*, *LAN discovery*, componentes para facilitar la serialización de objetos en Unity y dispone de una cantidad extensa de documentación de la API y ejemplos. Unity ofrece un servicio de pago para alojar servidores UNet junto con servidor maestro y de *matchmaking* y servidores *relay*.

- **Photon PUN[14]**

Solución *networking* estable que usa UDP y TCP, programada en C#. Implementa el soporte para establecer conexiones, un protocolo de comunicación con distintos tipos de calidad de servicio, servidor maestro y de *matchmaking*, RPC, componentes para facilitar la serialización de objetos en Unity y dispone de una cantidad extensa de documentación de la API, tutoriales y ejemplos.

Usa ENet como protocolo base (9). La empresa creadora de Photon actúa como servidor, por lo que la versión gratuita cuenta con límite de CCU. Es de pago[15] y se requiere de un pago recurrente a lo largo del tiempo para aumentar los límites de la versión gratuita. Photon ofrece un servicio de pago[16] para alojar servidores maestros y de *matchmaking* y servidores *relay* en servidores propios.

- **Photon BOLT[17]**

Solución *networking* estable sobre UDP, programada en C#. Implementa el soporte para establecer conexiones, un protocolo de comunicación con distintos tipos de calidad de servicio, servidor maestro y de *matchmaking* alojado por Photon o propio, RPC, componentes para facilitar la serialización de objetos en Unity haciendo un uso eficiente del ancho de banda y dispone de una cantidad extensa de documentación de la API y tutoriales. No dispone de una

[14] https://doc.photonengine.com/en-us/pun/current/getting-started/pun-intro
[15] https://www.photonengine.com/en-US/OnPremise/pricing
[16] https://www.photonengine.com/en-US/PUN/pricing
[17] https://www.photonengine.com/en-us/BOLT

versión gratuita y requiere pagar para aumentar el límite de CCU de forma recurrente a lo largo del tiempo[18].

- **Forge[19]**

Solución *networking* no estable sobre UDP, de código abierto y programada en C#. Implementa el soporte para establecer conexiones, un protocolo de comunicación con distintos tipos de calidad de servicio, *NAT hole punching*, servidor maestro y de *matchmaking*, *LAN discovery*, volver atrás en el tiempo, RPC y más funcionalidades. Dispone de documentación de la API además de vídeo tutoriales (10).

- **DarkRift Networking[20]**

Solución *networking* estable sobre UDP, programada en C#. Implementa el soporte para establecer conexiones, un protocolo de comunicación con distintos tipos de calidad de servicio, RPC, documentación de la API y ejemplos.

- **Lidgren**

Solución *networking* estable de código abierto, programada en C#. Implementa soporte para establecer conexiones, un protocolo de comunicación con distintos tipos de calidad de servicio, documentación de la API y ejemplos. No automatiza la sincronización del estado del videojuego entre los jugadores.

- **LiteNetLib**

Solución *networking* estable de código abierto, programada en C#. Implementa soporte para establecer conexiones, un protocolo de comunicación con distintos tipos de calidad de servicio, documentación de la API y ejemplos. No automatiza la sincronización del estado del videojuego entre los jugadores.

- **IceLink[21]**

Solución *networking* estable de código cerrado de pago, programada en C#. Es la única implementación de WebRTC en C# disponible. Implementa la API WebRTC por completo, que incluye dos calidades de servicio equivalentes a UDP y TCP, además de transmisión de audio y video. Cuesta un mínimo de 2,248 $. No automatiza la sincronización del estado del videojuego entre los jugadores.

- **WebRTC Network[22]**

Solución *networking* programado en código nativo compilado en librerías junto a un complemento programado en JavaScript que permite la conexión UDP usando WebRTC entre las plataformas Windows, Mac, Android y WebGL.

[18] https://www.photonengine.com/en-US/BOLT/pricing
[19] https://github.com/BeardedManStudios/ForgeNetworkingRemastered
[20] https://assetstore.unity.com/packages/tools/network/darkrift-networking-16711
[21] https://www.frozenmountain.com/products-services/icelink/
[22] https://assetstore.unity.com/packages/tools/network/webrtc-network-47846

	UNet	Photon PUN	Photon BOLT	Forge	DarkRift	Lidgren	LiteNetLib	IceLink	WebRTC Network
Estable	Sí	Sí	Sí	No	Sí	Sí	Sí	Sí	No
UDP	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
TCP	Sí	Sí	No	No	No	No	No	Sí	No
Cross-platform	Sí	Sí	Sí	No	No	No	No	Sí	No
WebRTC	No	No	No	No	No	No	No	Sí	Sí
C#	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No
Conexión	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Protocolo	Sí	Sí	Sí	Sí	Sí	Sí	Sí	No	No
Calidades	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
RPC	Sí	Sí	Sí	Sí	Sí	No	No	No	No
Host migration	Sí	No	No	No	No	No	No	No	No
LAN discovery	Sí	No	Sí	Sí	No	Sí	Sí	No	No
Componentes	Sí	Sí	Sí	No	No	No	No	No	No
Documentación	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Ejemplos	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí	Sí
Servidor maestro	Sí	Sí	Sí	No	No	No	No	Sí	Sí
Servidor *relay*	Sí	Sí	Sí	No	No	No	No	Sí	Sí
Versión Gratis	Sí	Sí	No	Sí	Sí	Sí	Sí	No	Sí

Tabla 1 Comparativa de las soluciones existentes.

2.1. Crítica al estado del arte

La generalidad es el mayor problema para las anteriores soluciones. Mientras que cumplen con el objetivo de permitir la realización de un videojuego en línea, limitan lo que es posible realizar en dicho videojuego.

Cada uno de los complementos mencionados anteriormente resuelve el problema de la sincronización del estado del videojuego entre los jugadores de modos distintos y dado que no es posible la sincronización perfecta en tiempo real, cada solución ofrece al menos una forma de realizar la sincronización, realizando un compromiso entre la calidad, precisión y completitud del estado mostrado y como de reciente es. La forma más adecuada de sincronizar el estado del videojuego depende en gran medida de la percepción del jugador, que es subjetivo, en ocasiones haciendo necesario implementar mecanismos de sincronización únicos para cada videojuego. La

sincronización del estado del juego usando las herramientas por defecto que ofrecen las anteriores soluciones suele presentar problemas, como parpadeos en los movimientos o aceleraciones en objetos que deberían de desplazarse a una velocidad constante y uniforme.

Debido a que la plataforma WebGL tiene restricciones de seguridad impuestas por los navegadores web, comunicaciones P2P entre navegadores únicamente son posibles mediante el uso de la API WebRTC, que además es el único modo de usar el protocolo UDP en un navegador web. Sin el uso de esta API también es posible comunicarse con otro navegador, pero es necesario el uso de un intermediario y del protocolo TCP, o bien es necesario usar extensiones en el navegador web. Solo la solución IceLink permite la comunicación UDP entre un navegador y cualquier otra plataforma soportada por Unity, pero al no automatizar la sincronización del estado del videojuego entre los jugadores, el desarrollador es responsable de integrar IceLink en otro sistema que lo use como un método de transporte. Las anteriores librerías no están diseñadas para soportar distintos o múltiples métodos de transporte y conexión entre usuarios, haciendo esta tarea complicada.

WebGL es la plataforma idónea para compartir un videojuego para ser testado debido a que los navegadores web son seguros y multiplataforma, aunque el rendimiento de un videojuego en un navegador web es menor (alrededor de 2 veces más lento[23]). Por ello, este TFG pretende cubrir la carencia de una solución completa (que ofrezca conexión entre jugadores, envío de datos entre ellos y automatización de la sincronización del estado del videojuego entre los jugadores) con soporte para todas las plataformas soportadas por Unity. Es importante indicar que, aunque WebRTC es un estándar, la implementación de los distintos navegadores existentes en el mercado no es idéntica y todavía se itera sobre el estándar realizando cambios.

Por último, existen plataformas como videoconsolas o Steamworks que ofrecen herramientas propietarias para la realización de conexiones entre jugadores, usualmente optativas, pero con múltiples convenientes ventajas. Estas herramientas se encargan de conectar a los jugadores y de enviar información entre ellos, pero funcionan mediante una capa de conexión independiente a la que usan las anteriores librerías. Algunos complementos ofrecen la opción de implementar capas de conexión y transporte distintas a las incluidas por defecto, pero es común encontrar dificultades a la hora de hacerlo ya que las nuevas requieren tener un funcionamiento y especificaciones similar a las incluidas por defecto, limitación que puede crear una incompatibilidad imposible de saltar con facilidad, como por ejemplo asumir que la capa de transporte se comunica directamente con un `Socket`.

2.2. Propuesta

Se proponen distintos puntos:

- Una solución capaz de funcionar en todas las plataformas soportadas por Unity (incluyendo WebGL) y facilitando la integración con distintas capas de transporte y conexión, como las ofrecidas por plataformas como videoconsolas, Steamworks o incluso TCP en lugar de UDP, mediante la separación de la lógica de establecimiento de

[23] https://docs.unity3d.com/es/current/Manual/webgl-performance.html

una conexión e intercambio de paquetes de la lógica usada para determinar el contenido de los paquetes. Con esta decisión se espera que, por ejemplo, juegos gratuitos que se juegan en el navegador sean más sencillos de desarrollar con funcionalidades en línea y sin limitar el videojuego a la plataforma web.

- Implementar el código completamente en C#, el lenguaje de scripting usado por Unity y que funciona en todas las plataformas que Unity soporta. Usar únicamente este lenguaje garantiza compatibilidad con todas las plataformas soportadas por Unity, pero a cambio no es posible obtener el mayor rendimiento posible de cada plataforma como podría realizarse al usar librerías en código nativo, una por cada plataforma.

- Implementaciones específicas para una plataforma pueden ser consideradas optimizaciones, siendo posible llevarlas a cabo cuando se considere necesario. Algunos de los anteriores complementos no están implementados en C#, en su lugar C# se usa como intermediario entre Unity y la implementación específica para la plataforma, dando lugar a incompatibilidades con plataformas que no cuentan con una implementación específica. Usar C# otorga el beneficio de soportar nuevas plataformas futuras, que inevitablemente aparecerán, como recientemente la videoconsola Switch de Nintendo o una posible futura PlayStation 5 de Sony.

- El uso de recursos realizado por los anteriores complementos tiene en común que el uso del ancho de banda puede ser reducido, en ocasiones a expensas de un mayor uso de otros recursos como memoria o CPU. En ninguno de ellos es posible indicar qué recurso es más importante en detrimento de los demás, por lo que se propone dar dicha opción permitiendo que, por ejemplo, videojuegos con gran consumo de memoria y CPU usen más ancho de banda mientras que videojuegos con gran cantidad de objetos sincronizados en línea, pero uso bajo de memoria y CPU, usen menos ancho de banda, sin necesidad de alterar el complemento o usar uno distinto.

- Las herramientas de sincronización dadas por los complementos existentes no ofrecen suficientes opciones de configuración por lo que, tan pronto se decide usar en un videojuego dichas herramientas incluidas por defecto y se encuentra que no son adecuadas por cualquier razón, es necesario implementar código propio que podría no haber sido necesario de tener más opciones de configuración. Se propone la elaboración de un complemento más configurable.

- Uso de estándares siempre que sea posible. WebRTC hace uso de estándares que se pueden usar en otras implementaciones, como STUN para *NAT hole punching* o TURN para servidores *relay* entre jugadores que no pueden establecer una conexión directa entre ellos. Los anteriores complementos prácticamente no hacen uso de estándares y en su lugar implementan funcionalidades desde cero, limitando en ocasiones opciones futuras como expansión del videojuego mediante alquiler de servidores o incrementando costes de desarrollo al no poder hacer uso de herramientas existentes que sí hacen uso de estándares.

- Por último, se asume que un desarrollador necesitará implementar funcionalidad en línea que no está programada dentro del complemento, por lo que se pretende facilitar la

programación de dichas funcionalidades sin quitar control u ocultando funcionalidad disponible en el mismo.

3. Análisis del problema

El listado de problemas concretos existentes en todos los complementos listados anteriormente es el siguiente:

- Elevado uso de ancho de banda, CPU y/o memoria de las soluciones existentes. A mayor uso de recursos, menor debe ser la complejidad del estado que debe de compartirse entre los jugadores para posibilitar que el videojuego funcione.
- Imposibilidad de establecer una conexión usando UDP entre la plataforma WebGL en un navegador web y el resto de las plataformas soportadas por Unity.
- La plataforma WebGL no puede iniciar un servidor que acepte conexiones desde plataformas distintas a WebGL, y en caso de poder sólo se puede comunicar con determinadas plataformas.
- Integrar plataformas como Steamworks o videoconsolas es complicado, en ocasiones imposible.
- Limitada selección de métodos de sincronización del estado del videojuego, con limitaciones o problemas para el juego objetivo, que dan lugar a la necesidad de implementar desde cero técnicas de sincronización en cada juego, ignorando los elementos incluidos.

3.1. Análisis de seguridad

Un complemento para Unity que permite la conexión de jugadores en línea permite la comunicación remota entre máquinas conectadas a internet. Cualquier fragmento de código dedicado al procesamiento de datos recibidos a través de internet es susceptible a ser atacado remotamente, dando lugar a un problema de seguridad. Los videojuegos en línea suelen ser un objetivo para *hackers* por lo extendidos que están, haciendo sencillo encontrar una máquina ejecutando el videojuego, siendo posible usar los servidores maestros del propio videojuego para encontrar dichas máquinas. La seguridad no siempre es una prioridad cuando se implementan soluciones para crear videojuegos en línea, el objetivo suele estar centrado en rendimiento y compresión de datos, dando lugar a gran cantidad de casos límite en el código que tienen como resultado vulnerabilidades que pueden ser explotadas (11) (12) (13).

Otro problema es el de jugadores que quieren hacer trampas. En algunos casos estos jugadores pueden realizar acciones que van desde problemas no relacionados con la seguridad, como romper reglas del juego, hasta problemas de seguridad, como hacerse pasar por otro jugador conectado a un servidor e intentar provocar su desconexión o ataques de denegación de servicio (DOS) a clientes o servidores (14), entre otros.

Además, información del juego puede filtrarse a terceros si la conexión es interceptada o si existe un intermediario en la conexión, pero se desconoce su existencia (ataque *man-in-the-middle*). Esta información puede ser mensajes de texto o audio enviados por el sistema de mensajería del juego o el estado del videojuego.

Por estas razones, toda información recibida desde internet debe de tratarse como un potencial ataque y debe considerarse en todo momento como insegura. La implementación del código junto con pruebas unitarias exhaustivas es una medida recomendable. Se debe esperar que

eventualmente se recibirá un ataque y por ello se debe de estar preparado a ellos. Por último, el desarrollador del videojuego debe poder elegir opciones de seguridad, como si se desea encriptar la información intercambiada entre los jugadores, o incluso dar a los jugadores la opción de elegir.

3.2. Identificación y análisis de las soluciones posibles

Dado el anterior listado de problemas, se presenta a continuación posibles soluciones para cada uno de ellos:

- **Elevado uso de ancho de banda, CPU y/o memoria de las soluciones existentes.**
    - Usar un complemento existente con un bajo uso de recursos.
    - Optimizar el uso de recursos de un complemento existente.
    - Implementar un nuevo complemento que haga un uso de recursos bajo.
- **Imposibilidad de establecer una conexión usando UDP entre la plataforma WebGL en un navegador web y el resto de las plataformas soportadas por Unity.**
    - Usar una librería existente que implemente WebRTC en todas las plataformas soportadas por Unity.
    - Implementar una librería que implemente WebRTC en todas las plataformas soportadas por Unity.
- **La plataforma WebGL no puede iniciar un servidor que acepte conexiones desde plataformas distintas a WebGL, y en caso de poder sólo se puede comunicar con determinadas plataformas.**
    - Añadir la opción de usar WebRTC como medio de conexión.
- **Integrar plataformas como Steamworks o videoconsolas es complicado, en ocasiones imposible.**
    - Permitir métodos de transporte de datos arbitrarios que funcionen con el complemento.
- **Limitada selección de métodos de sincronización del estado del videojuego, con limitaciones o problemas para el juego objetivo, que dan lugar a la necesidad de implementar desde cero técnicas de sincronización en cada juego, ignorando los elementos incluidos.**
    - Copiar métodos de sincronización existentes de otras librerías cuya licencia lo permita.
    - Implementar nuevos métodos de sincronización.

A continuación se detalla el listado de soluciones indicando las ventajas, desventajas y viabilidad de cada una de ellas:

- **Usar un complemento existente con un bajo uso de recursos.**
    - Ventajas:
        - Ahorro considerable de tiempo en la implementación de código.
        - Partir de una implementación estable.
        - El desarrollador y la comunidad de usuarios pueden ayudar cuando surgen problemas.

        - El creador del complemento es responsable de su correcto funcionamiento y mantenimiento.
    - Desventajas:
        - Limitaciones del complemento requieren reescribir código. Dependiendo de la limitación puede ser necesario rehacer todo el complemento.
        - Necesidad de invertir dinero en el complemento si no es gratuito (cada uno presenta una modalidad de pago distinta, desde pagos únicos a pagos mensuales dependiendo de la cantidad de jugadores conectados simultáneamente).
    - Viabilidad:
        - Los complementos existentes se suelen poder dividir en capas, una de conexión y transferencia de datos y otra de escritura y lectura de datos. Sería necesario rehacer la capa de conexión y transferencia de datos en los complementos existentes para soportar WebRTC, Steamworks y videoconsolas. Este cambio puede suponer la reescritura entera de una capa y en ocasiones parte de la otra en caso de no estar implementado de forma modular. No es posible conocer la viabilidad sin analizar el código del complemento, que en caso de los que son de pago requeriría su compra.
- **Optimizar el uso de recursos de un complemento existente.**
    - Ventajas:
        - Ahorro considerable de tiempo en la implementación de código.
        - Partir de una implementación estable.
    - Desventajas:
        - Alterar el complemento termina con la estabilidad original.
        - El desarrollador y la comunidad no pueden ayudar en caso de que surjan problemas
        - A partir del momento en el que se modifica el código del complemento se pierde la posibilidad de actualizarlo con nuevos cambios, convirtiéndose en nuestro propio complemento a la hora de mantenerlo.
        - Necesidad de invertir dinero en el complemento si no es gratuito (cada uno presenta una modalidad de pago distinta, desde pagos únicos a pagos mensuales dependiendo de la cantidad de jugadores conectados simultáneamente).
    - Viabilidad:
        - Al asumir que se realizarán cambios en partes críticas del complemento asumimos que necesitamos una comprensión elevada sobre las partes internas del complemento. El tiempo requerido para comprender y alterar la implementación original y así optimizar el uso de recursos puede requerir un tiempo mayor que el que costaría implementar complemento nuevo. La viabilidad de la anterior solución es aplicable a esta solución.
- **Implementar un nuevo complemento que haga un uso de recursos bajo.**
    - Ventajas:
        - Código a medida.
        - Sólo las funcionalidades necesarias.
    - Desventajas:
        - Alto coste temporal.
        - Alta probabilidad de introducir errores en el código que puedan pasar inadvertidos.

        - No se puede contar con la comunidad de usuarios con la que cuenta un complemento existente para conseguir ayuda.
    - Viabilidad:
        - Las dos anteriores soluciones requerirían la implementación de una cantidad de código nuevo similar a la necesaria en esta solución, considerando a dichas soluciones comparadas con esta, por tanto, redundantes. Dado que la realización de un trabajo de fin de grado requiere de una inversión de tiempo elevada, se puede considerar que el alto coste temporal es un requisito inevitable del trabajo por lo que no es tan desventajoso.
- **Usar una librería existente que implemente WebRTC en todas las plataformas soportadas por Unity.**
    - Ventajas:
        - Ahorro considerable de tiempo en la implementación de código sin importar si se usa un complemento existente como base o se crea uno nuevo.
        - Mantenimiento de la librería corre a cargo de sus creadores.
        - Dado que una librería suele abarcar un rango muy específico de funcionalidades son implementadas como un módulo, normalmente haciendo sencilla su inclusión en código existente.
        - Dado que la API WebRTC requiere del uso de encriptación, una librería existente es muy probable que haga un uso optimizado de los recursos para esta tarea, probablemente usando a su vez otra librería como OpenSSL o mbedTLS[24].
    - Desventajas:
        - Si la librería carece del rendimiento necesario sería necesario optimizarla. Dependiendo de la librería, dicha tarea puede suponer pequeños o grandes cambios en el código.
        - WebRTC tiene limitaciones e inconvenientes en el protocolo que sería conveniente poder cambiar.
    - Viabilidad:
        - Las únicas librerías que hay están escritas en el lenguaje de programación C. Todas ellas tienen algún problema: la librería ya no está mantenida, no soporta ARM, no está disponible para todos los sistemas operativos que Unity soporta (sobre todo consolas), entre otras. Es una opción viable, pero compleja, y la falta de experiencia con el lenguaje C supone un problema.
- **Implementar una librería que incorpore WebRTC en todas las plataformas soportadas por Unity.**
    - Ventajas:
        - Poder esquivar problemas de la API WebRTC como no poder elegir el puerto de conexión o la necesidad de usar un puesto por cada conexión. El diseño de WebRTC no contempla la existencia de servidores dedicados, no presentando por tanto funcionalidades útiles para ellos.

[24] https://tls.mbed.org/

        - Implementar la librería en C# permitiría que funcione en cualquier plataforma soportada por Unity ya que este se encarga de generar el ejecutable.
    - Desventajas:
        - Alto coste temporal.
        - Protocolo complejo que requiere hacer uso de encriptación.
        - Una librería implementada en C# puede tener un rendimiento inferior al de librerías existentes.
    - Viabilidad:
        - El autor ha dedicado tiempo en el pasado en la implementación de una librería en C# que permita hacer uso de la API WebRTC. Gracias a ello dispone de experiencia para realizar esta tarea. En caso de tener una implementación en C# que funcione en todas las plataformas es posible añadir optimizaciones mediante el uso de librerías existentes específicas para distintas arquitecturas de CPU y sistemas operativos.
- **Añadir la opción de usar WebRTC como medio de conexión.**
    - Ventajas:
        - Conexión posible entre todas las plataformas de Unity.
    - Desventajas:
        - Mayor complejidad del complemento.
    - Viabilidad:
        - La implementación de alguna de las anteriores dos soluciones implementaría esta solución de forma automática.
- **Permitir métodos de transporte de datos arbitrarios que funcionen con el complemento.**
    - Ventajas:
        - Facilidad de inclusión de nuevos métodos de transporte como los usados por Steamworks o la videoconsola PlayStation 4.
    - Desventajas:
        - Necesario reescribir código en todos los complementos existentes (de usar uno). Si se va a implementar un nuevo complemento, el diseño del mismo puede contemplar este hecho.
    - Viabilidad:
        - Sencillo de implementar en un complemento nuevo, pero complejo en complementos existentes que están ligados al método (o métodos) de transporte por defecto. En caso de que el complemento ya soporte varios métodos de transporte es posible que esta tarea sea sencilla.
- **Copiar métodos de sincronización existentes de otras librerías con licencia que así lo permitan.**
    - Ventajas:
        - Un método de sincronización suele ser algo específico, normalmente la implementación de un algoritmo, que es sencillo de copiar.
        - A más métodos de sincronización, más opciones para el desarrollador del videojuego y más probable que encuentre uno que sea adecuado para el videojuego que desarrolla.
    - Desventajas:
        - -
    - Viabilidad:

        - Debería ser sencillo llevar la tarea a cabo. Los métodos de sincronización son la implementación de algoritmos que suelen estar explicados en internet, en ocasiones junto con una implementación en pseudocódigo o algún lenguaje de programación, dando la opción de implementarlo con relativa facilidad en caso de no poderse copiar.
- **Implementar nuevos métodos de sincronización.**
    - Ventajas:
        - A más métodos de sincronización, más opciones para el desarrollador del videojuego y más probable que encuentre uno que sea adecuado para el videojuego que desarrolla.
    - Desventajas:
        - Requiere investigación.
    - Viabilidad:
        - En todos los complementos existentes añadir nuevos métodos de sincronización es algo sencillo, estando diseñados precisamente con esta tarea como uno de sus objetivos, permitiendo el envío y recepción arbitrario de datos, que es el único requisito para realizar esta tarea.

A partir del anterior análisis se ha decidido llevar a cabo las siguientes soluciones, teniendo en cuenta que, debido a que se está realizando un trabajo de fin de grado, se dispone de una gran cantidad de tiempo y es, por tanto, una oportunidad para optar por soluciones que debido a la inversión de tiempo requerida podrían no haber sido rentables:

- Implementar un nuevo complemento que haga un uso de recursos uso de recursos bajo.
- Implementar una librería que implemente WebRTC en todas las plataformas soportadas por Unity.
- Añadir la opción de usar WebRTC como medio de conexión.
- Permitir métodos de transporte de datos arbitrarios que funcionen con el complemento.
- Copiar métodos de sincronización existentes de otras librerías con licencia que así lo permitan.
- Implementar métodos de sincronización nuevos.

3.3. Solución propuesta

La solución propuesta consiste en la implementación de un nuevo complemento para Unity para la realización de videojuegos en línea. Este complemento debe de realizarse planeando que use la menor cantidad de recursos posibles, que sea modular, que soporte distintos métodos de transporte y conexión como WebRTC y Steamworks, que incluya métodos de sincronización de estado de otras librerías y que permita incluir métodos nuevos.

El trabajo se divide en las siguientes fases, en las cuales también se dividirá el TFG:

1. **Implementación de una capa de transporte y conexión modular que permita la inclusión sencilla de nuevas capas de transporte y conexión.**
    1.1. Capa de conexión directa sobre UDP.
    1.2. Capa de conexión mediante WebRTC.

1.2.1. Implementación de una librería WebRTC.

2. **Implementación de un protocolo que añada funcionalidad sobre las anteriores capas y que esté orientado a las necesidades de un videojuego.**
   2.1. Comunicación mediante segmentación del búfer de datos usando identificadores numéricos coincidentes entre ambos jugadores.
   2.2. Métodos de sincronización con distintas calidades de servicio:
      2.2.1. Con pérdidas desordenado: Sin garantías de recepción u orden, equivalente al protocolo UDP.
      2.2.2. Sin pérdidas ordenado: Garantía de recepción de toda la información enviada y en el orden enviado.
      2.2.3. Con pérdidas ordenado: Garantía de recepción de al menos la última información enviada y en orden.
   2.3. Envío de información en búfers de tamaño arbitrario.
   2.4. Envío de información encapsulada en estructuras.
      2.4.1. Compresión de estructuras mediante compresión diferencial.
   2.5. Priorización, para que en casos en los que sea necesario fragmentar la información antes de transmitirla sea posible priorizar el envío de unos datos frente a otros.
3. **Implementación de métodos de sincronización del estado del videojuego.**
   3.1. Sincronización de `Transform` en Unity.
   3.2. Sincronización de `Animator` en Unity.
4. **MVP 1.**
   4.1. Documentación del complemento.
   4.2. Prueba de uso por testadores.
   4.3. Iterar cambios sobre lo realizado en los puntos 2 y 3 conforme a la realimentación proporcionada por los testadores.
5. **MVP 2.**
   5.1. Documentación del complemento.
   5.2. Prueba de uso por testadores.
   5.3. Iterar cambios sobre lo realizado en los puntos 2 y 3 conforme a la realimentación proporcionada por los testadores.
6. **Documentación del complemento.**

Todas las fases relacionadas con la implementación del código deben de realizarse al mismo tiempo que se añaden test unitarios para garantizar, en la medida de lo posible, el funcionamiento deseado.

4. Evaluación de la idea de negocio

4.1. Estudio de mercado

4.1.1. Clientes

Todos aquellos que quieren hacer un videojuego en línea en Unity son clientes objetivo. Asset Store, la tienda oficial de Unity, es la mejor fuente de datos para conocer la cantidad de clientes potenciales, pero no permite ver la cantidad de unidades vendidas de cada producto. Afortunadamente la tienda permite publicar comentarios a los usuarios de cada producto en la página web de dicho producto y en el caso de productos de pago, únicamente aquellos que han comprado el producto pueden comentar en él. Es necesario tener en cuenta que para poder comentar en la página de la tienda de un complemento hay que descargarlo antes, siendo necesario comprarlo si fuera de pago, que no todos los compradores de un complemento comentan en el mismo y que la Asset Store muestra la cantidad de tiempo que ha pasado desde que se publicó un comentario en lugar de la fecha concreta del envío, mostrando cantidades como "Hace un año" o "Hace tres meses", haciendo imposible conocer exactamente en qué año fue publicado el comentario.

Para extraer los datos se ha creado el script *asset-store-parser.php* con el lenguaje de programación PHP. Se puede encontrar en el punto Anexos.

En la figura 1 se muestran los datos recopilados, a fecha 1 de agosto, mediante el recuento de la cantidad de usuarios únicos que han dejado al menos un comentario en alguno de los siguientes complementos temática similar:

- Atavism 2018.2 OP Standard
- DarkRift Extreme
- DarkRift Networking
- DarkRift Networking 2
- DarkRift Networking 2 - Pro
- DarkRift Pro
- Forge Networking Remastered
- Master Server Framework
- MultiLan - Multiplayer network kit
- MultiOnline - Multiplayer online kit
- NAT Traversal
- NetDrone Unity
- Network Sync Transform (NST)
- Networking and Serialization Tools (TNet 3)
- Photon Bolt
- Photon PUN+
- Photon Unity Networking Free
- Photon Viking Multiplayer Showcase
- Smooth Sync
- TinyBirdNet

- WebRTC Network
- WebRTC Video Chat

De los datos extraídos se puede apreciar que la cantidad de clientes objetivo aumenta cada año y que en los últimos 7 años la cantidad de clientes total es de al menos casi 500 personas.
Teniendo en cuenta las limitaciones de los datos recopilados, se puede asumir que la cantidad de clientes potenciales es mayor que la aquí representada.

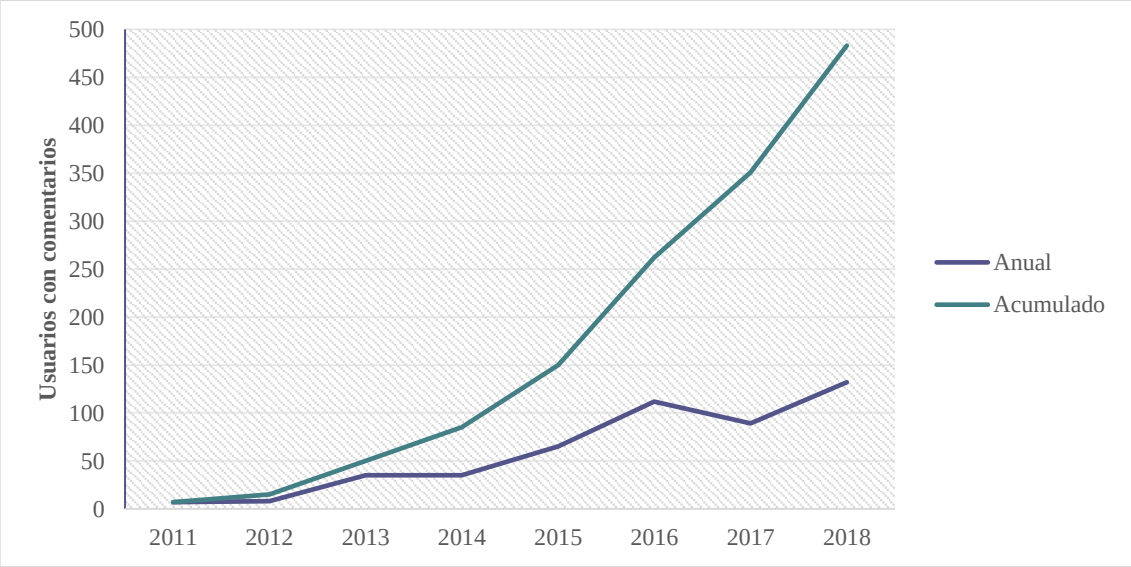

Figura 1 Usuarios con comentarios en complementos relevantes de la tienda de Unity.

4.1.2. Productos competidores

Se ha considerado como producto competidor aquellos que ofrecen todo lo necesario para realizar un videojuego online en Unity, descartando aquellos que únicamente ofrecen una abstracción para el establecimiento de conexiones junto con varias calidades de transmisión. A continuación se encuentra listado cada complemento junto con sus características:

UNet

- **Enlace**:
  - Integrado en Unity.
  - https://bitbucket.org/Unity-Technologies/networking
- **Alojamiento**: Por jugadores y en servidores propios.
- **Servidor maestro**: Ofrecido por Unity o alojado en servidor propio.
- **Código**: Sincronización de variables, escritura en búfer, *host migration*, LOD por distancia al jugador, RPC, modo desconectado y encriptación.
- **Conectividad**: Conexión directa, *LAN discovery* y servidores *relay* de Unity.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Todas las plataformas de Unity. WebGL limitada a WebSocket (TCP).
- **Arquitectura**: Cliente-servidor, no autoritativo y P2P.
- **Coste**: Gratis. El uso de los servidores *relay* cuesta 0.4 €/GB.

WebRTC Network

- **Enlace**:
  - https://assetstore.unity.com/packages/tools/network/webrtc-network-47846
  - https://assetstore.unity.com/packages/tools/network/webrtc-video-chat-68030
- **Alojamiento**: Por jugadores y en servidores propios.
- **Servidor maestro**: En servidor propio.
- **Código**: Escritura en búfer.
- **Conectividad**: Conexión directa y servidores *relay* que usen protocolo TURN.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Windows, Mac, Linux y WebGL.
- **Arquitectura**: P2P.
- **Coste**: Gratis. Soporte para transmisión de audio y vídeo cuesta 84,87 €.

DarkRift Networking 2

- **Enlace**:
  - https://assetstore.unity.com/packages/tools/network/darkrift-networking-2-95309
  - https://assetstore.unity.com/packages/tools/network/darkrift-networking-2-pro-95399
- **Alojamiento**: Por jugadores y en servidores propios.
- **Servidor maestro**: No incluido.
- **Código**: Escritura en búfer y mensajería mediante Identificadores.
- **Conectividad**: Conexión directa.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Todas las plataformas de Unity menos WebGL.
- **Arquitectura**: Cliente-servidor, múltiples servidores, P2P, no autoritativo y autoritativo.
- **Coste**: Gratis. Características extras como soporte para *plugins* en el servidor y filtro de palabras ofensivas por 89,33 €.

Photon Unity Networking

- **Enlace**:
  - https://assetstore.unity.com/packages/tools/network/photon-unity-networking-free-1786
  - https://assetstore.unity.com/packages/tools/network/photon-pun-12080
- **Alojamiento**: En servidores propios y servidores alojados por Photon.
- **Servidor maestro**: Ofrecido por Photon o alojado en servidor propio.
- **Código**: Escritura en búfer, LOD por publicación/suscripción, RPC, modo desconectado y encriptación.
- **Conectividad**: Mediante los servidores *relay* de Photon.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Todas las plataformas de Unity. WebGL limitada a WebSocket (TCP).
- **Arquitectura**: Cliente-servidor y no autoritativo.
- **Coste**:
  - Servidores de Photon:
    3GB por CCU.
    20 CCU gratis.
    100 CCU 84,87 €/5 años.

100 CCU 15,70 €/mes.
500 CCU 84,87 €/mes.
1000 CCU 152 €/mes.
500 mensajes/s por *room*[25].

- Servidor propio:
  500 CCU 84,87 €/mes.
  más de 500 CCU 152 €/mes.

Photon Bolt

- **Enlace**:
  - https://assetstore.unity.com/packages/tools/network/photon-bolt-83233
- **Alojamiento**: Por jugadores y en servidores propios.
- **Servidor maestro**: Ofrecido por Photon o alojado en servidores propios.
- **Código**: Escritura en búfer, compresión de datos, *lag compensation*, LOD por priorización y objetivos, RPC, modo desconectado y encriptación.
- **Conectividad**: Conexión directa, *NAT hole punching*, *LAN discovery* y servidores *relay* de Photon.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Todas las plataformas de Unity menos WebGL y Steamworks.
- **Arquitectura**: Cliente-servidor, no autoritativo y P2P.
- **Coste**:
  - Precio base 84.87 €.
  - Servidores de Photon:
    3GB per CCU.
    20 CCU gratis.
    100 CCU gratis 5 años.
    100 CCU 15,70 €/mes.
    500 CCU 84,87 €/mes.
    1000 CCU 152 €/mes.
    500 mensajes /s por r*oom*.
  - Servidor propio:
    Gratis.

Forge Networking Remastered

- **Enlace**:
  - https://assetstore.unity.com/packages/tools/network/forge-networking-remastered-38344
- **Alojamiento**: Por jugadores y en servidores propios.
- **Servidor maestro**: En servidor propio.
- **Código**: Escritura en búfer, compresión de datos, *lag compensation*, LOD, RPC y modo desconectado.
- **Conectividad**: Conexión directa, *NAT hole punching* y *LAN discovery*.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Todas las plataformas de Unity menos WebGL y Steamworks.

[25] Photon considera *room* a un espacio virtual compartido por varios jugadores.

- **Arquitectura**: Cliente-servidor, no autoritativo, autoritativo y P2P.
- **Coste**: Gratis.

MultiOnline - Multiplayer online kit

- **Enlace**:
    - https://assetstore.unity.com/packages/tools/network/multionline-multiplayer-online-kit-10663
- **Alojamiento**: Por jugadores y en servidor propio.
- **Servidor maestro**: En servidor propio.
- **Código**: Escritura en búfer, RPC y modo desconectado.
- **Conectividad**: Conexión directa y *LAN discovery*.
- **Protocolos**: UDP.
- **Plataformas**: Todas las plataformas de Unity menos WebGL. Necesaria una versión de Unity inferior a 2018.
- **Arquitectura**: Cliente-servidor, no autoritativo y P2P.
- **Coste**: 40,20 €.

NetDrone Unity

- **Enlace**:
    - https://assetstore.unity.com/packages/tools/network/netdrone-unity-51175
- **Alojamiento**: Por jugadores y en servidores propios.
- **Servidor maestro**: En servidor propio.
- **Código**: Escritura en búfer, RPC y encriptación.
- **Conectividad**: Conexión directa.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Todas las plataformas de Unity. WebGL limitada a WebSocket (TCP).
- **Arquitectura**: Cliente-servidor, no autoritativo y P2P.
- **Coste**: 71,47 €.

Networking and Serialization Tools (TNet 3)

- **Enlace**:
    - https://assetstore.unity.com/packages/tools/network/networking-and-serialization-tools-tnet-3-56798
- **Alojamiento**: Por jugadores y en servidores propios.
- **Servidor maestro**: En servidor propio.
- **Código**: Escritura en búfer, LOD mediante publicación/suscripción, RPC, guardar partida automatizado y subir/descargar archivos en servidor.
- **Conectividad**: Conexión directa y *LAN discovery*.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Todas las plataformas de Unity menos WebGL y Steamworks.
- **Arquitectura**: Cliente-servidor, no autoritativo y P2P.
- **Coste**: 84,87 €.

TinyBirdNet

- **Enlace**:
    - https://assetstore.unity.com/packages/tools/network/tinybirdnet-110128

- **Alojamiento**: Por jugadores y en servidores propios.
- **Servidor maestro**: No incluido.
- **Código**: Sincronización de variables, escritura en búfer y RPC.
- **Conectividad**: Conexión directa.
- **Protocolos**: UDP.
- **Plataformas**: Todas las plataformas de Unity menos WebGL.
- **Arquitectura**: Cliente-servidor, no autoritativo y P2P.
- **Coste**:
    - Uso no comercial: Gratis.
    - Uso comercial: 26,80 €.

Atavism 2018.2 OP Standard

- **Enlace**:
    - https://assetstore.unity.com/packages/tools/network/atavism-2018-2-op-standard-117342
- **Alojamiento**: Servidor propio.
- **Servidor maestro**: Servidor propio.
- **Código**: Scripts en servidor y escritura en búfer dependiendo de los scripts del servidor.
- **Conectividad**: Conexión directa.
- **Protocolos**: TCP y UDP.
- **Plataformas**: Todas las plataformas de Unity menos WebGL.
- **Arquitectura**: Cliente-servidor y autoritativo.
- **Coste**:
    - 20 CCU por 445,77 €.
    - 500 CCU por 862 €.
    - 3.000 CCU por 4.314 €.
    - sin límite CCU por 8.630 €.

El producto desarrollado para este TFG tendría las siguientes características:

AshNet

- **Alojamiento**: Servidor propio.
- **Servidor maestro**: Servidor propio.
- **Código**: Escritura en búfer, RPC, Sistema de mensajería por identificadores, encriptación, compresión de datos, LOD y modo desconectado.
- **Conectividad**: Conexión directa y usando servidores *relay* TURN.
- **Protocolos**: UDP.
- **Plataformas**: Todas las plataformas de Unity.
- **Arquitectura**: Cliente-servidor y no autoritativo.
- **Coste**: 100 €.

Con este producto se mejoraría el código frente a los competidores al enfatizar una mayor variedad de características, entre ellas la compresión de datos. Además, el producto contaría con la característica exclusiva de soportar todas las plataformas de Unity usando el protocolo UDP.

4.2. Análisis DAFO

	Internas	Externas
Debilidades	- Carencia de escenarios de prueba reales. - Falta de experiencia en juegos con muchos jugadores en línea. - Seguridad.	- Gran cantidad de competencia variada. - Unity está trabajando en un nuevo plugin junto con Google. - Complementos con herramientas de pago.
Fortalezas	- Experiencia con otros complementos. - Experiencia en realizar un complemento del mismo tipo. - Avances lentos en la conexión a internet, poca probabilidad de quedar desactualizados. - Uso de estándares.	- Cada videojuego online tiene necesidades distintas. - Ningún complemento soporta todas las plataformas de Unity, suele faltar soporte para WebGL. - WebGL está en alza. - Cada vez más creadores de videojuegos.

4.3. Modelo de negocio y proyección económica a 3 años

El proyecto será llevado a cabo por 2 personas en un local alquilado. El coste mensual por persona se estima en 1000 € y el de las instalaciones en 500 €. Además, se requiere una inversión inicial de 1000 € en equipo informático que se espera que dure al menos 3 años.

El complemento AshNet se venderá por 100 € en la tienda Asset Store de Unity, que se queda con un 25%, dejando un total de 75 € por unidad vendida. En el inicio la cantidad de ventas esperadas es de alrededor de 10 copias mensuales, pero se espera que pasado medio año aumenten a 15, terminando por caer gradualmente conforme pasen los meses. Para atraer mercado se liberará el código fuente bajo la licencia Creative Commons BY-NC-SA[26] que permite la copia, redistribución y adaptación de la obra bajo la obligación de dar crédito, no usar el con propósitos comerciales y compartir con la misma licencia que la original.

Este complemento será realizado como parte de un videojuego que se venderá en el futuro y que requerirá de la implementación de otros nuevos complementos. Se espera que dichos complementos sean desarrollados pasando alrededor de 6 meses entre cada uno. Estos nuevos complementos serán puestos a la venta por un precio similar y se espera que sigan una progresión de ventas similar al primer complemento.

[26] https://creativecommons.org/licenses/by-nc-sa/4.0/

	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
Ingresos												
AshNet	2.250 €	2.250 €	3.375 €	3.375 €	2.250 €	1.125 €	1.125 €	1.125 €	750 €	450 €	450 €	375 €
complem. 2			2.250 €	2.250 €	3.375 €	3.375 €	2.250 €	1.125 €	1.125 €	1.125 €	750 €	450 €
complem. 3					2.250 €	2.250 €	3.375 €	3.375 €	2.250 €	1.125 €	1.125 €	1.125 €
complem. 4							2.250 €	2.250 €	3.375 €	3.375 €	2.250 €	1.125 €
complem. 5									2.250 €	2.250 €	3.375 €	3.375 €
complem. 6											2.250 €	2.250 €
Costos												
Personal	6.000 €	6.000 €	6.000 €	6.000 €	6.000 €	6.000 €	6.000 €	6.000 €	6.000 €	6.000 €	6.000 €	6.000 €
Equipo infor.	2.000 €											
Instalaciones	1.500 €	1.500 €	1.500 €	1.500 €	1.500 €	1.500 €	1.500 €	1.500 €	1.500 €	1.500 €	1.500 €	1.500 €
Beneficios												
Beneficios	-7.250 €	-12.500 €	-14.375 €	-16.250 €	-15.875 €	-16.625 €	-15.125 €	-14.750 €	-12.500 €	-11.675 €	-8.975 €	-7.775 €

4.4. Lean Canvas

2 **Problemas**	**4** **Solución**	**3** **Proposición de valor**	**9** **Ventaja competitiva**	**1** **Clientes**
Es complejo realizar un videojuego *online*. Ninguna solución actual permite el uso del protocolo UDP en navegadores web, únicamente TCP (juegos *online* en tiempo real usando TCP sufren de pausas periódicas cuya frecuencia depende de la calidad de la conexión a internet). Las alternativas actuales o bien hacen un uso	-Sencillez para realizar un videojuego *online*. -Soporte del protocolo UDP (WebRTC) para navegadores web (ninguna solución lo tiene). -Reducido uso del ancho de banda (pocas lo hacen). -Modularidad para poder añadir funcionalidad y código simple y documentado (pocas lo hacen).	Realizar videojuegos *online* para navegadores web es conveniente, pero todos los productos actuales que ofrecen soporte lo hacen esperando condiciones de internet ideales (sin pérdida de paquetes), en caso contrario la experiencia de juego estará llena de interrupciones.	Se espera que otras librerías no incluyan próximamente UDP para navegadores web debido a su complejidad que supone y que no lo han hecho hasta ahora. La sencillez de uso del complemento y bajo uso de ancho de banda son difíciles de copiar.	Aquellos que desean realizar una aplicación online en tiempo real con el motor de videojuegos Unity. Se conocen personas cercanas que están interesadas en una solución alternativa a las actuales. Existe la posibilidad de tener clientes que no usen el

<table>
<tr>
<td>excesivo del ancho de banda (limitan lo que se puede hacer en un videojuego online) o carecen de ciertas características que no pueden ser añadidas sin reingeniería.</td>
<td>8 Métricas

-Cantidad de interactuación en el foro.
-Adopción a través de la tienda.</td>
<td></td>
<td>5 Canales

Se publicitará el producto en los foros oficiales del motor Unity y se distribuirá usando su tienda de complementos.</td>
<td>motor Unity, pero usen el lenguaje C# para realizar aplicaciones.</td>
</tr>
<tr>
<td colspan="3">7 Costes

Fijos mensuales: 2000 € (personal) + 500 € (instalaciones)
Variables: 2000 € (Equipos informáticos)</td>
<td colspan="2">6 Ingresos

Venta de copias del plugin y futuros plugin. Ingresos de 75 € por copia vendida, requiere vender un mínimo de 34 copias mensuales para obtener beneficio.</td>
</tr>
</table>

4.5. Conclusiones de la evaluación

Todos los datos extraídos para este análisis no son reales, son una estimación propia que puede o no tener correlación con los datos reales por falta de acceso a estos. Además, este complemento está enfocado para Unity pero podría orientarse a únicamente C#, abriendo la puerta a más mercado. Google Trends[27] muestra, como se puede ver en la figura 2, que hay interés por "WebRTC" y también por "Unity WebGL", mostrando tras compararlas que la cantidad de potenciales clientes puede ser mucho mayor que la aquí mostrada.

De la proyección económica se observa que los beneficios empezarán a crecer a partir del primer año, pero a un ritmo lento, tardando 3 años en volver a la situación inicial. Para ser viable económicamente sería necesario realizar cambios, como por ejemplo crear nuevos complementos cada 3 meses en lugar de cada 6, pero para ello podría hacer falta más personal. Con los datos de los que se dispone y especulando en la cantidad de clientes potenciales usando una estimación propia, se podría esperar más ventas de las que se han calculado anteriormente, haciendo de esta una idea de negocio viable.

[27] https://trends.google.com/trends/

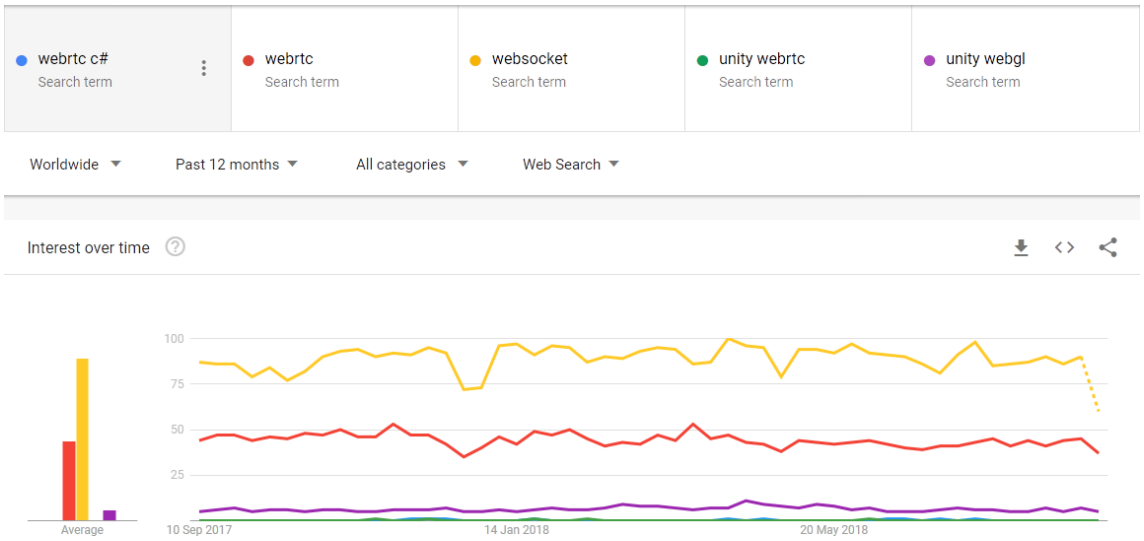

Figura 2 Resultados de Google Trends para WebRTC, Unity u WebSocket.

5. Diseño de la solución

5.1. Arquitectura del sistema

La arquitectura del sistema está inspirada en el modelo OSI que estructura la lógica de un protocolo de comunicación en múltiples capas. El complemento estará compuesto por tres capas que se situarán encima de un protocolo de comunicación orientado a conexión que permita enviar y recibir datos. Estas capas serán:

5.1.1. Capa de transporte y conexión

Esta capa abstraerá el proceso de conexión con otros jugadores de manera que los detalles específicos de cada método de transporte sean independientes de la capa de mensajería, que únicamente necesita poder enviar y recibir datos. La capa implementará protocolos de conexión usando UDP y WebRTC y dará la opción de añadir otros protocolos, como Steamworks o protocolos usados por videoconsolas como PlayStation4 y Xbox one (estos protocolos e información relacionada con el desarrollo para videoconsolas no están abiertos al público, son mantenidos en secreto por los desarrolladores aceptados en las plataformas y no es posible implementar estos métodos de transporte sin antes ser aceptado por los fabricantes).

5.1.2. Capa de mensajería

Se encargará de enviar y recibir datos entre los jugadores, entre ellos la información necesaria para sincronizar el reloj de los jugadores. Se usará un sistema de mensajería basada en identificadores, en la que cada mensajero contará con una prioridad dinámica e intercambiará datos con el otro jugador. Esta capa proveerá de acuses de recibo a los mensajeros para que conozcan si la información que transmiten alcanza al otro jugador y decidan qué hacer en consecuencia, implementando así distintos niveles de calidad de conexión, como sin pérdidas y ordenado (se reenviarán datos no recibidos por el otro jugador, que los interpretará en el mismo orden que en el que se enviaron) o con pérdidas y ordenado (no reenviará datos perdidos a no ser que el que se pierda sea el último que se ha enviado).

La idea del uso de mensajería es novel frente a la competencia. El resto de los complementos presentan una cantidad configurable de canales con una calidad de conexión configurable. Un canal es equivalente a un mensajero, con un identificador y datos que enviar, pero a diferencia de un mensajero el canal no puede ser enviado junto con otros canales, requiriendo un envío usando la capa de transporte por cada canal que necesita enviar datos. Dado que en dichos canales capas superiores escribirán datos que muy probablemente usan un identificador, El mensajero tiene la intención de unificar el identificador del canal con el identificador del objeto de la capa superior, granulando así los canales y reduciendo la cantidad de envíos necesarios a expensas de hacer envíos probablemente más pesados.

5.1.3. Capa de Unity

Tendrá como objetivo administrar todo lo relacionado con sincronizar el estado del videojuego entre los jugadores, usando las herramientas proveídas por las dos capas anteriores. Esta capa

necesita poder establecer y terminar conexiones con otros jugadores a petición del jugador, para ello haciendo uso de la primera capa, además de usar la segunda capa para sincronizar el estado del videojuego. Se pretende seguir un camino similar a los complementos competidores, ofreciendo scripts que permiten sincronizar el estado de objetos individuales entre los jugadores simplemente añadiendo el script al objeto, y que el desarrollador pueda configurar el script mediante una serie de opciones como compresión o prioridad de dicho objeto frente a otros para ser sincronizado.

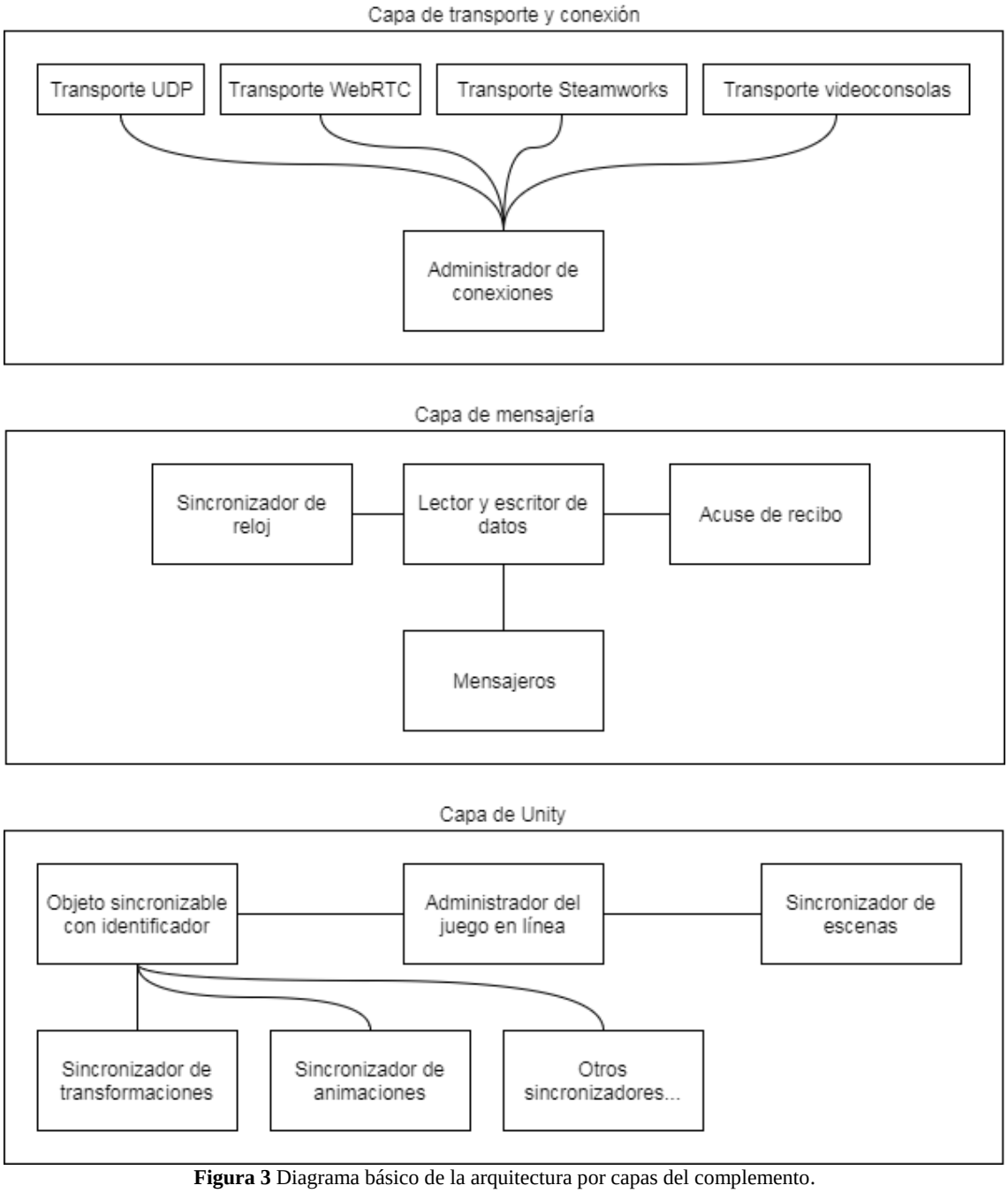

Figura 3 Diagrama básico de la arquitectura por capas del complemento.

5.2. Diseño detallado

5.2.1. Capa de transporte

El diseño de la capa de transporte se encuentra representado en la figura 4. Nótese que está relacionada con la figura 5.

`NetManager` es la clase principal. Administra multiples `socket` para conectarse a otros `NetManager` remotos usando `SocketsController` y tiene *callbacks* que se ejecutan cuando se realiza una conexión o desconexión. `IPeer` representa una conexión remota y permite enviar y recibir paquetes. Un `NetManager` se inicializa con un conjunto de `INetGate` que contienen la funcionalidad necesaria para establecer una conexión a un `INetGate` equivalente de un `NetManager` remoto. Una vez un `INetGate` consigue establecer una conexión, crea un `IPeer` que contiene dicha conexión y se lo pasa a *NetManager* para que lo notifique como conectado, invocando un *callback*, `IPeer` es el encargado de finalizar su propia conexión y notificarlo a `NetManager` para que invoque el *callback* pertinente. Los *callbacks* de `NetManager` tienen la intención de permitir la ejecución de lógica ante una conexión o desconexión. En el caso de WebRTC, tanto `GateWebRtc` como `PeerWebRtc` hacen uso de la API completa y sus protocolos. El diseño pretende que `NetManager` no se preocupe por la implementación de `INetGate` o `IPeer`, permitiendo extender el complemento para soportar otros tipos de conexiones además de las ya presentes. Esta capa es independiente de Unity.

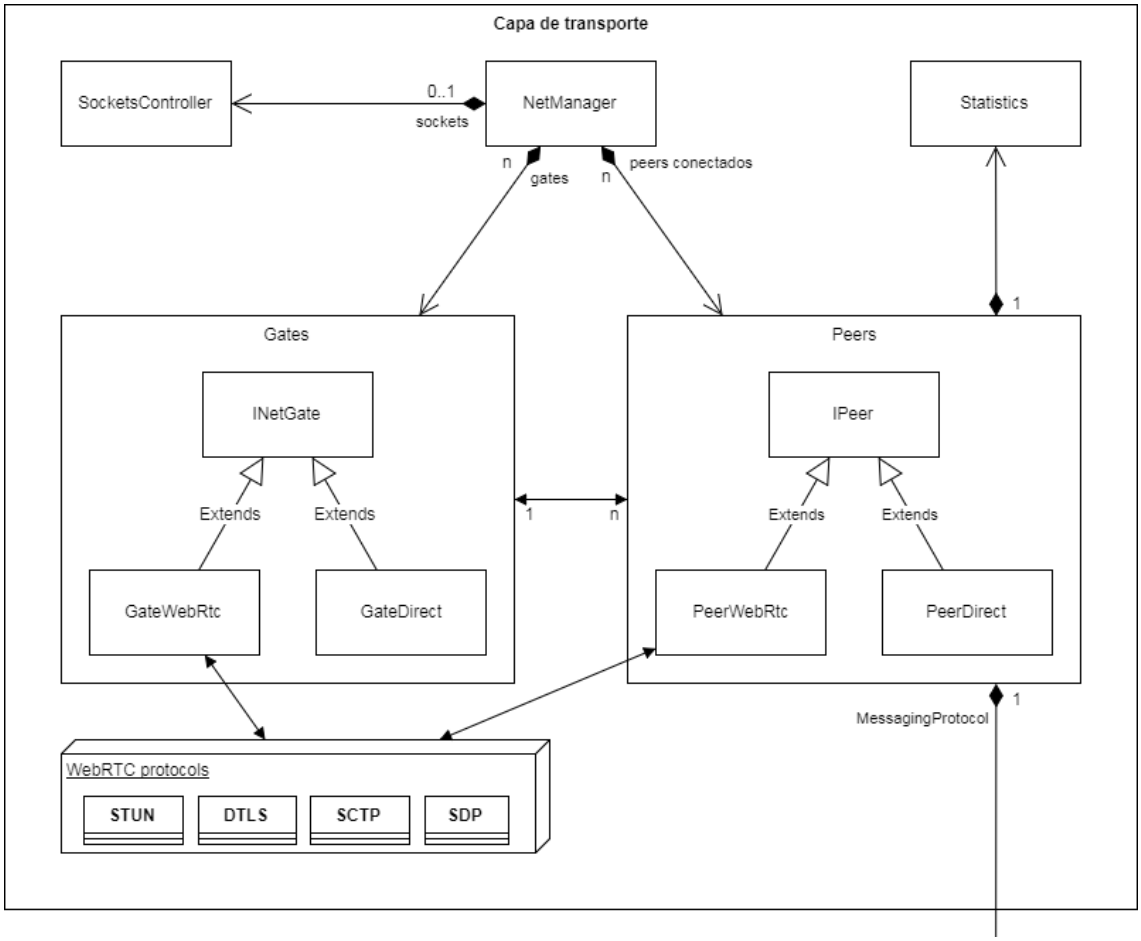

Figura 4 Diseño detallado de la capa de transporte.

5.2.2. Capa de mensajería

El diseño de la capa de mensajería se encuentra representado en la figura 5. Nótese que está relacionada con las figuras 4 y 6.

Un `IPeer` tiene un `PeerMessenger` que hace uso del protocolo que es muestra en la figura 7. `RemoteTimeTracker` hace seguimiento del reloj remoto para conocer su valor. `AckManager` hace un seguimiento de los paquetes para conocer cuáles se han perdido (si es que se ha perdido alguno) y notificar a los mensajeros de ello usando la interfaz `IAckReceiver`. Un `PeerMessenger` tiene múltiples `IMessenger` suscritos mediante un identificador que debe coincidir remotamente, pero es responsabilidad del que realiza las suscripciones de los mensajeros que estos tengan un identificador correcto.

Un `IMessenger` implementa las interfaces `ISender`, que se encargan de comprobar si el mensajero necesita enviar algo y qué prioridad tiene el envío que quiere hacer, e `IReceiver`, que se encarga de recibir mensajes remotos. Se han implementado dos tipos de mensajeros: uno de estructuras serializables que implementan la interfaz `IPacket` (`MessengerPacketReliableOrdered` y `MessengerPacketLatestState`) y uno de búfers que pueden ser fragmentados si no caben en un paquete cuando van a ser enviados (`MessengerBufferReliableOrdered`).

La interfaz `IMessengerCreator` crea un `IMessenger` para que múltiples `IPeer` tengan `IMessenger` equivalentes. `NetRPC` envía mensajes entre múltiples `IPeer` en modo cliente-servidor.

Esta capa es independiente de Unity.

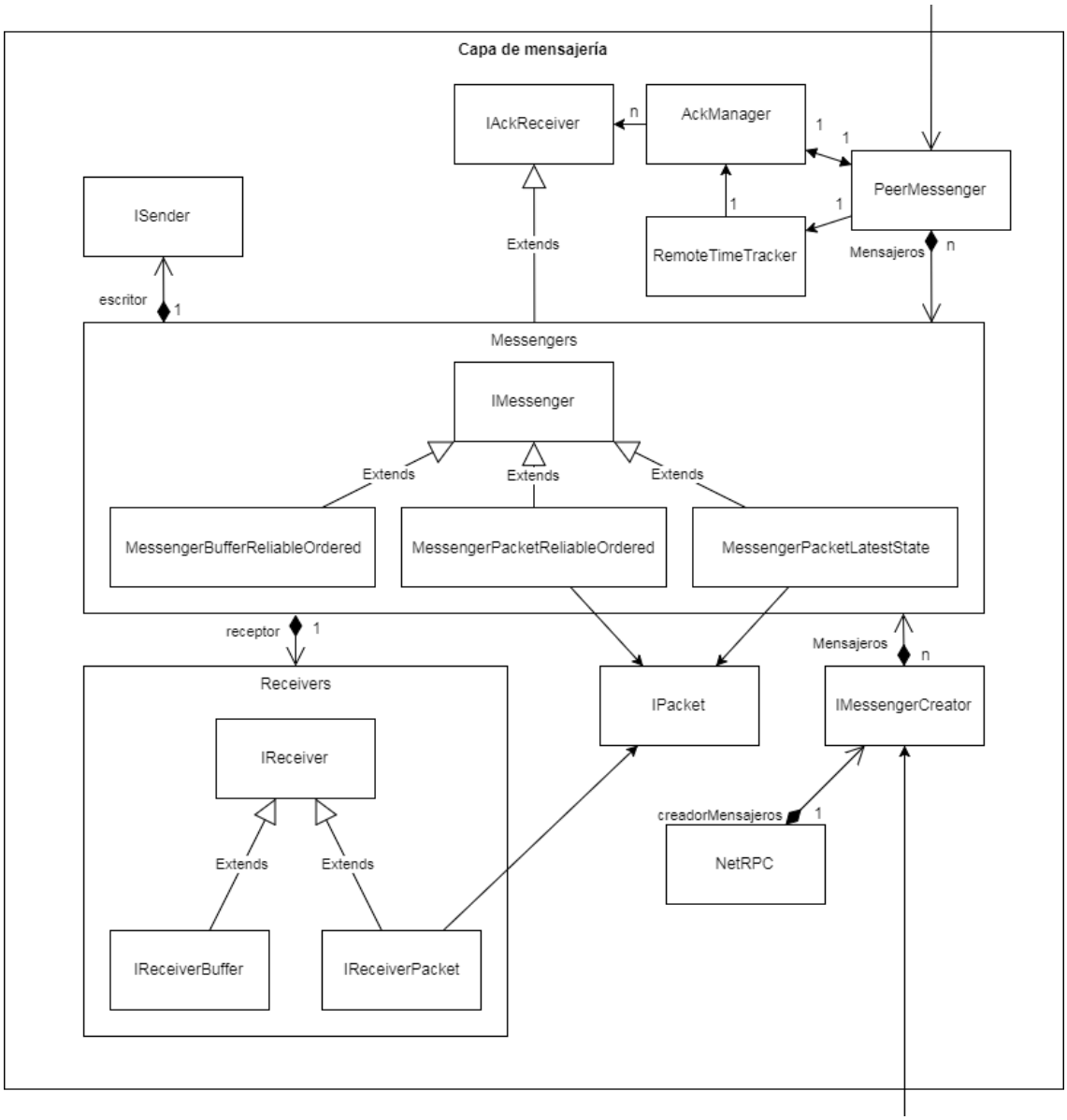

Figura 5 Diseño detallado de la capa de mensajería.

5.2.3. Capa de Unity

El diseño de la capa de mensajería se encuentra representado en la figura 6. Nótese que está relacionada con la figura 5.

El componente de Unity *NetController* controla los componentes *NetClient* y *NetServer* unificando la lógica de un *NetManager* con los *INetGate* del tipo *GateDirect* y *GateWebRTC*. Los *IPeer* que se conectan son envueltos con un objeto NetHost que los abstrae como una instancia de Unity conectada en la que pueden estar jugando múltiples personas, representadas por los objetos *NetPlayer*.

NetIdentity, que puede o no tener a un *NetPlayer* como propietario, representa a un actor en línea o parte de este (un *NetIdentity* puede encontrarse dentro de otro *NetIdentity* en la jerarquía de la escena), provee a un *GameObject* de un identificador y

garantiza que todas las instancias de Unity conectadas lo comparten. `NetIdentity` contiene un lisado de objetos que implementen la interfaz `INetInit` y los inicia cuando se establece una conexión para iniciar la lógica de la sincronización del estado mediante componentes como `NetAnimator` o `NetTransform`.

`GateWebGL` y `PeerWebGL` son variantes de `GateWebRtc` y `PeerWebRtc` para la plataforma WebGL, implementadas con un plugin escrito en JavaScript, y que son funcionalmente idénticas. `AshNetMaster` es una clase estática usada para simplificar el uso del servidor maestro para establecer conexiones usando `SignalingDirect` y `SignalingWebRtc`, que se comunican con `GateDirect` y `GateWebRtc` respectivamente.

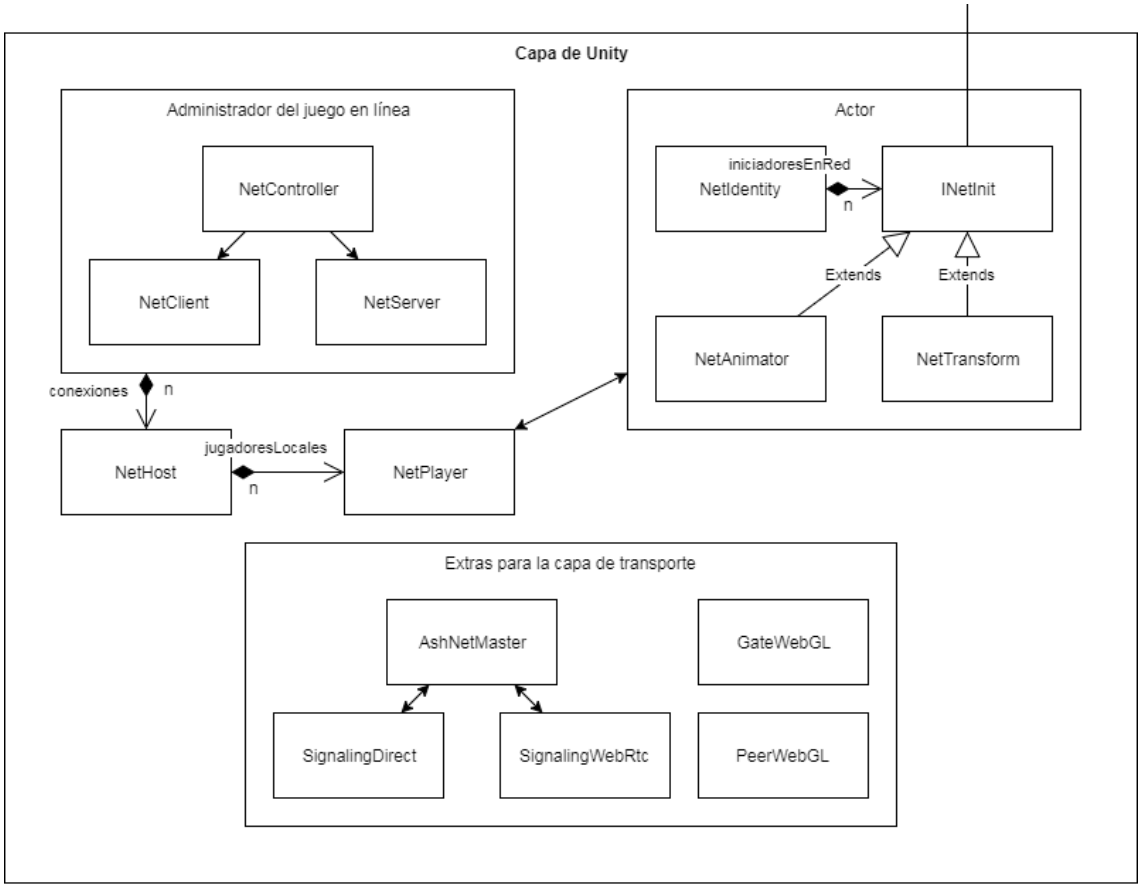

Figura 6 Diseño detallado de la capa de Unity.

5.2.4. Protocolo de mensajería

El diseño de la capa de mensajería se encuentra representado en la figura 7.

Un `PeerMessenger` ordena los mensajeros suscritos al mismo por orden de prioridad de forma descendente. Si algún mensajero necesita enviar un mensaje se inicia un paquete con las cabeceras necesarias y los mensajeros, por orden de prioridad, escriben en el paquete hasta llenarlo.

Las cabeceras son escritas por `AckManager` y `RemoteTimeTracker`. Cada mensajero escribe su identificador único y la cantidad de bits que el mensaje ocupa, junto con el mensaje.

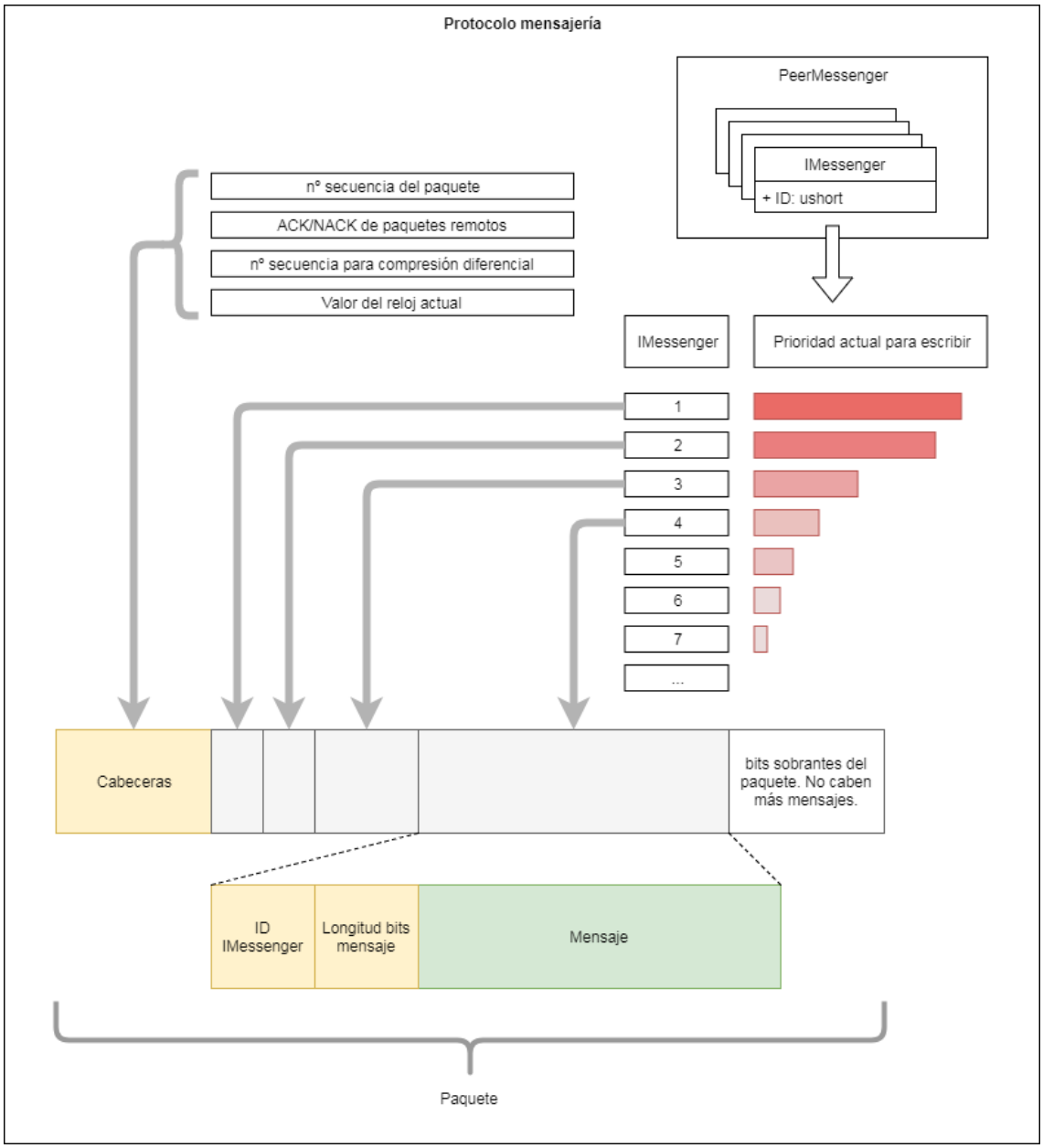

Figura 7 Diseño detallado del protocolo de mensajería.

6. Tecnologías utilizadas

6.1. Unity

Unity[28] es el motor de videojuegos para el que se realizará el complemento. Este motor permite crear videojuegos y aplicaciones interactivas para múltiples plataformas (Windows, Mac, Linux, Android, iOS, Windows Phone, Web, PlayStation 4, Xbox one, PlayStation Vita, Nintendo Wii U y Nintendo Switch entre otras). Es posible crear tanto aplicaciones con gráficos en dos dimensiones como en tres, reproducir sonido, usar distintos periféricos de entrada como mandos y más. El motor cuenta con una gran comunidad de usuarios activos que aportan valor al motor, la mayoría de la cual se puede encontrar en el foro oficial, además de una tienda virtual en la que se vende contenido variado realizado por los propios usuarios. La aplicación es de pago bajo una suscripción mensual, pero se puede usar de forma gratuita siempre que los beneficios obtenidos con su uso no superen los 100.000 € anuales.

El motor permite ser extendido para añadir funcionalidad tanto al juego como a la interfaz del editor (la ventana con la que se interactúa cuando se usa Unity) mediante el uso del lenguaje de programación C#, creando scripts. Gracias a esto no sólo es posible alterar la lógica del videojuego sino también extender la interfaz sin límites, permitiendo adaptar el motor a cualquier necesidad y extenderlo hasta convertirlo incluso en un programa distinto.

La elección del motor se debe a la grata experiencia previa con el mismo. Este complemento para crear videojuegos en línea pretende cubrir la necesidad que ha surgido durante el desarrollo de un videojuego usando Unity junto con los integrantes de un equipo de desarrollo. Las razones que han llevado a esta necesidad se encuentran explicadas en el punto Análisis del problema.

[28] https://unity3d.com/

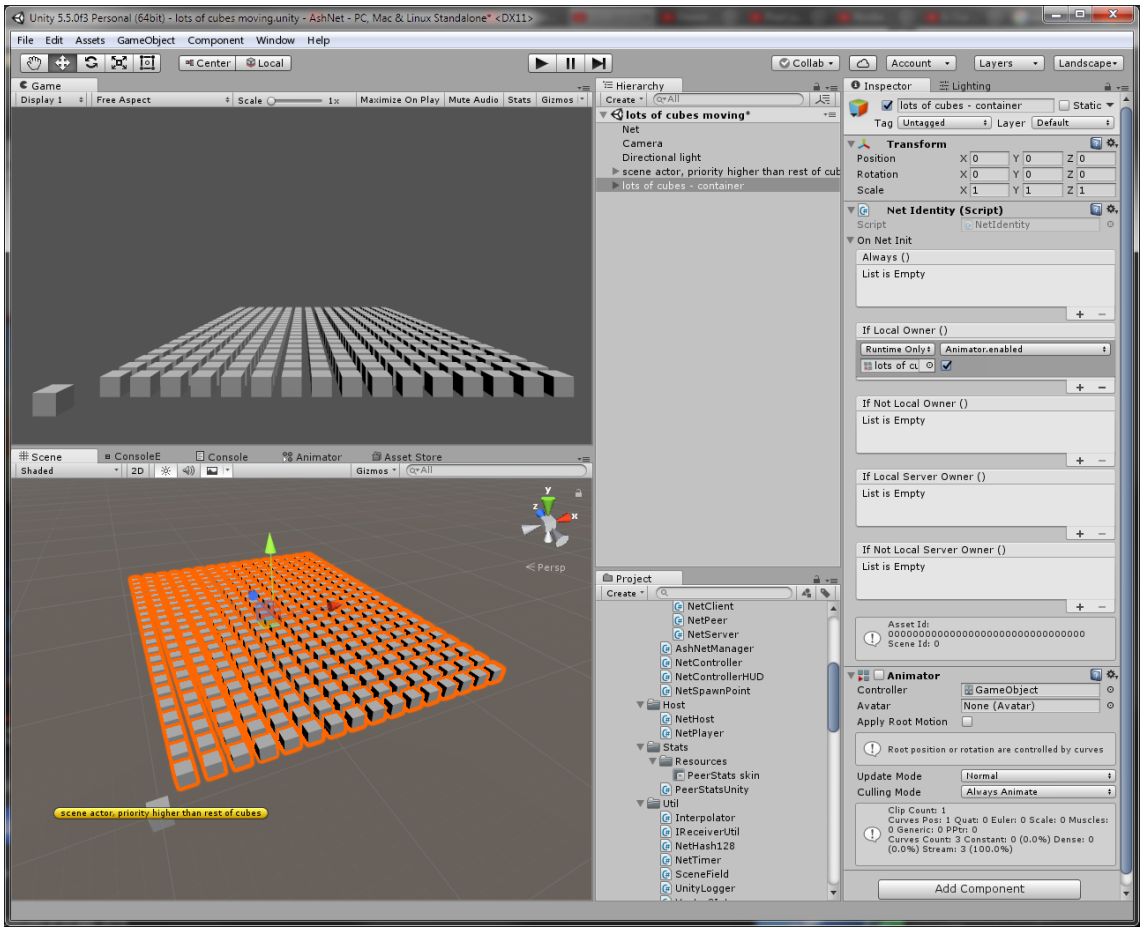

Figura 8 Captura de pantalla de la aplicación Unity con el proyecto AshNet abierto.

6.2. Visual Studio

Visual Studio[29] es un IDE (entorno de desarrollo integrado) que permite editar código C#, siendo el editor más conocido y usado para este lenguaje de programación. Tiene gran cantidad de herramientas que ayudan en la edición del código como autocompletar, formato automático del código, seguimiento de referencias, detección de diversos errores en el código antes de compilar, recomendaciones para arreglar problemas comunes y más. Cuenta con tres herramientas que han ayudado notablemente al desarrollo del complemento: depurador, ventana inmediata, perfilador y explorador de pruebas unitarias.

- El depurador permite ejecutar el código paso a paso mediante el uso de puntos de interrupción que, tras ponerlos sobre una línea del código, pausan la ejecución del código cuando esta alcanza el punto de interrupción. Estos puntos son configurables y permitiendo el uso de condiciones escritas con el mismo lenguaje que el código que está siendo iterado (en este caso C#), aunque añadir una condición a un punto de interrupción disminuye notablemente la velocidad de ejecución del programa. Una vez interrumpido el código es posible observar el valor de variables, alterarlas e incluso alterar el código ejecutado sin recompilar y ejecutar el programa de nuevo. En caso de

[29] https://visualstudio.microsoft.com/

que existan varios hilos, una vez pausada la aplicación es posible comprobar dónde se ha detenido la ejecución en dichos hilos e incluso continuar el proceso de depuración en ellos.

- La ventana inmediata permite ejecutar código durante una sesión de depuración mientras la ejecución esté pausada. Gracias a esta herramienta es posible cambiar el valor de variables, ejecutar métodos, crear objetos y comprobar el valor de cualquier variable que sea posible acceder desde el ámbito de ejecución actual. En caso de provocar una excepción mientras se usa esta ventana, si esta no es capturada por el código se propagará a la ventana inmediata sin detener la aplicación.
- El perfilador permite analizar el rendimiento de distintas partes del código a lo largo del tiempo, concretamente el uso de la CPU, memoria y problemas de paralelismo con hilos que bloquean a otros hilos. Aunque las mediciones del perfilador son una aproximación (debido, entre otras razones, a que el perfilador necesita alterar la ejecución de la aplicación para poder realizar las mediciones) y por sí mismo no es capaz de desvelar todos los problemas de rendimiento presentes, sí puede dar pistas o apuntar a problemas, siendo de gran utilidad.
- El explorador de pruebas unitarias permite ejecutar pruebas unitarias con gran facilidad. Tras la ejecución de una prueba unitaria permite conocer el tiempo transcurrido durante su ejecución, si ha fallado y dónde, y la salida que se generó durante la ejecución.

Esta herramienta se ha usado para generar las librerías que Unity reconoce. Estas librerías están compuestas por código CIL (Infraestructura de lenguaje común) y no por código nativo. Unity, durante el proceso de compilación, incorpora este código junto con el resto de código escrito en scripts en C# dentro del motor, permitiendo que la librería funcione en todas las plataformas soportadas por Unity (exceptuando el uso de las API no soportadas por plataformas determinadas).

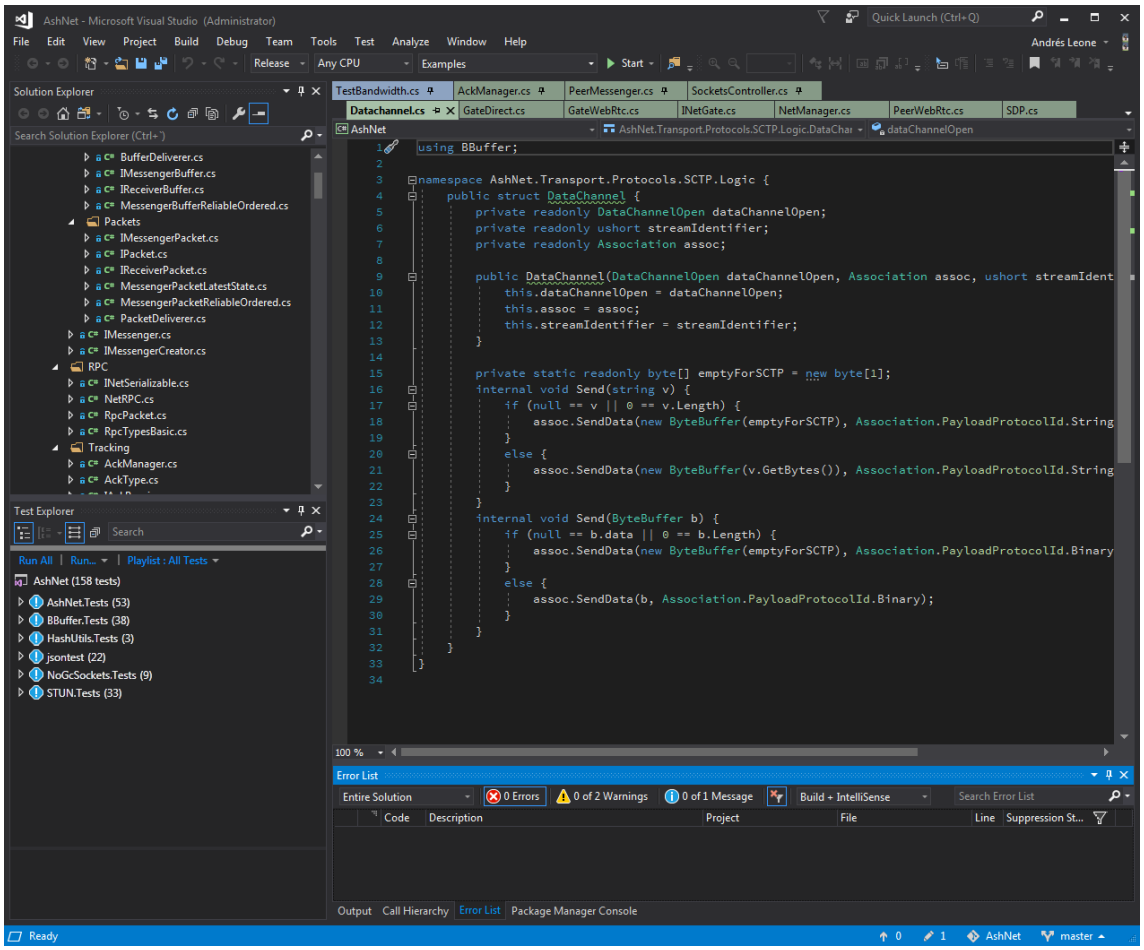

Figura 9 Captura de pantalla de la aplicación Visual Studio con un proyecto abierto.

6.3. GIT

GIT es un controlador de versiones descentralizado que permite versionar archivos y coordinar trabajo entre múltiples personas. Es software libre bajo licencia GNU GPLv2 yestá diseñado para gestionar las versiones de los archivos en ramas cuyo historial puede ser explorado. Cuando múltiples personas trabajan en el mismo proyecto versionado por GIT, para sincronizar el trabajo se fusionan ramas y resuelven los conflictos que puedan surgir. Debido a que es descentralizado y capaz de funcionar sin un servidor e incluso sin conexión a internet, es de los controladores de versiones más populares, sino el más popular, con multitud de webs importantes[30] [31] [32] que permiten alojar repositorios GIT, muchas veces de forma gratuita (aunque con limitaciones).

GIT ha sido usado extensivamente en la realización del complemento, permitiendo trabajar en múltiples ordenadores, ayudando en la resolución de errores, permitiendo deshacer cambios que de otro modo habría sido imposible y facilitando compartir el código del proyecto con los participantes de los experimentos que se explican más adelante.

[30] http://github.com/
[31] https://bitbucket.org/product
[32] https://about.gitlab.com/

6.4. C#

C# es un lenguaje de programación orientado a objetos mediante el uso de clases, fuertemente tipado, imperativo, declarativo, funcional y genérico. Programas escritos en C# se compilan al lenguaje intermedio CIL que es después ejecutado por una máquina virtual CLR (Entorno en tiempo de ejecución de lenguaje común) que se encuentra dentro del paquete de software .Net Framework. El lenguaje fue desarrollado por Microsoft y cuenta con una estandarización ECMA e ISO (aunque las versiones más recientes del lenguaje no están estandarizadas de este modo). Microsoft únicamente desarrolló la máquina virtual que ejecuta el código CIL para el sistema operativo Windows por lo que, con el objetivo de implementar la máquina virtual y el compilador de C# en otros sistemas operativos, nació el proyecto de código abierto Mono[33]. Debido a que únicamente Mono permite ejecutar C# sin usar Windows, Unity lo usa en lugar de .Net Framework. A partir del año 2014 Microsoft publicó la plataforma llamada .NET Core como un proyecto de código abierto, cubriendo parte del espacio que Mono cubría hasta entonces al ofrecer soporte para sistemas operativos distintos a Windows, además de que el proyecto está publicado bajo la permisiva licencia MIT. En la actualidad el lenguaje se encuentra en la versión 7.3.

C# es el único lenguaje de programación soportado por Unity que, al estar implementado usando una versión antigua y alterada de Mono, sólo soporta la versión 3.5 del lenguaje. Unity soporta experimentalmente desde la versión 2018.1 la versión 6 del lenguaje C# y han anunciado que la versión 2018.3 soportará la versión 7.3.

Algunas características del lenguaje no soportadas por Unity pueden ser usadas en una librería en lugar de en scripts directamente en Unity. Esto es posible cuando el lenguaje intermedio generado es compatible con versiones anteriores.

6.5. JavaScript

JavaScript es un lenguaje de programación interpretado orientado a objetos, imperativo, débilmente tipado y dinámico que se desarrolló para ser ejecutado en navegadores web, siendo el lenguaje de programación por defecto en todos los navegadores y la única opción en la mayoría de ellos, además de ser principalmente usado en ese ámbito.

La plataforma WebGL de Unity compila el proyecto de modo que un navegador web moderno pueda ejecutarlo. En la actualidad Unity genera código JavaScript para esta plataforma, pero en versiones más modernas soporta WebAssembly, que ofrece un mayor rendimiento que usando JavaScript y menor espacio en disco para alojar el código compilado. Para conseguir que esta plataforma funcionara con el complemento ha sido necesario añadir una librería en JavaScript que pueda interactuar con el resto del código C# y viceversa, para poder hacer uso de las API WebRTC y WebSocket ya que ambas requieren del uso de la API `Socket`, que no está disponible en la plataforma WebGL.

[33] https://www.mono-project.com/

6.6. UDP

UDP (Protocolo de datagramas de usuario) es un protocolo del nivel de transporte en el modelo OSI que permite el envío y recepción de datagramas a través de la red sin necesidad del establecimiento de una conexión previa y sin necesidad de mantener un estado. El protocolo únicamente garantiza que se intentará entregar los datagramas a su destinatario, pero cuantas veces llegará el datagrama al destino (cero o más) o en qué orden llegarán. Además, el protocolo no informa al emisor si el datagrama ha llegado a su destino.

Este protocolo es de gran utilidad precisamente gracias a su escasez de garantías frente a la alternativa de usar el protocolo TCP, ya que menos garantías también supone menos restricciones. El complemento está orientado a la sincronización en tiempo real del estado de un videojuego, tarea que se beneficia de poder descartar el reenvío de información perdida y antigua porque en su lugar puede enviar información más reciente. Esta opción no presente en TCP, que en caso de un paquete perdido decide esperar a que sea reenviado. Sólo esta razón hace muy conveniente el uso de UDP, pero al no presentar garantías es necesario implementar un protocolo sobre UDP que ofrezca las garantías necesarias, en ocasiones implementando un protocolo similar a TCP.

UDP no está soportado por toda la red de internet, a diferencia de TCP. Por ejemplo, redes corporativas o escuelas pueden tener el protocolo UDP deshabilitado. Aunque esto supone un problema, este hecho no es común. Además, como no es un protocolo orientado a conexión, el puerto de salida de un datagrama en una puerta de enlace NAT puede cambiar con el tiempo, problema que afectó en el desarrollo de DOOM, tal y como relata John Carmack en su blog *"[…] and the router apears to be making the incorrect assumption that UDP is only used for simple request / response protocols. The router changes the UDP port while you are playing."*[34]. Este tipo de problemas añaden dificultades a la hora de implementar un protocolo sobre UDP.

6.7. WebRTC

WebRTC es una API que tiene como objetivo permitir realizar llamadas de voz y vídeo usando una conexión P2P, además de permitir transferir datos entre navegadores web, todo usando únicamente JavaScript. Esta API está disponible en los navegadores más recientes (Google Chrome, Mozilla Firefox, Opera, Microsoft Edge, Vivaldi y Safari entre otros) así como en las distintas plataformas en las que estos navegadores están presentes (Windows, Mac, Linux, Android, iOS y Tizen entre otros). WebRTC hace uso de los protocolos ICE, STUN, DTLS, SCTP y Datachannel y hace uso de UDP y TCP indistintamente, aunque para este complemento sólo se ha implementado WebRTC sobre UDP.

Debido a que WebRTC está implementado en navegadores web, la seguridad de la API es de gran importancia. El protocolo DTLS que encripta la comunicación está implementado de modo que obliga a elegir entre un listado limitado de algoritmos de encriptación, entre los cuales no se

[34] http://www.team5150.com/~andrew/carmack/johnc_plan_1997.html#d19971231 (fuente original no accesible)
http://web.archive.org/web/20120403213402/http://www.team5150.com/~andrew/carmack/johnc_plan_1997.html#d19971231)

encuentran algoritmos nulos (que no realizan ninguna encriptación). Esto supone la ventaja de seguridad por efecto, pero un inconveniente al afectar a la latencia.

WebRTC implementa STUN + DTLS multiplexado sobre TCP/UDP. Sobre DTLS se encuentra SCTP, que a su vez sobre él se encuentra Datachannel. A la hora de enviar y recibir datos se interactúa con Datachannel, que añade una cabecera a la información y se la pasa a SCTP, que a su vez añade una cabecera y se la pasa a DTLS que también añade una cabecera, encripta el contenido y finalmente se pasa a la capa de transporte TCP/UDP. A la hora de recibir un datagrama se comprueba si es STUN o DTLS mediante la comprobación de los primeros bytes y haciendo uso de que STUN escribe determinados bytes siempre con el mismo valor. Tras decidir que se trata de DTLS se elimina la cabecera DTLS y se desencripta, se pasa a SCTP que elimina su cabecera, se pasa a Datachannel que elimina su cabecera y finalmente pasa a la aplicación los datos que se enviaron originalmente.

Debido al uso de DTLS, SCTP y Datachannel, el número de bytes usados en un datagrama debido a las cabeceras es elevado, superior a 48 bytes. Este número situado en perspectiva frente al límite recomendado de bytes por datagrama, que es de 1200 bytes, supone un 4 % de la cantidad de bytes disponibles. Hay que recordar también que, aunque el tope sea 1200 bytes, a menor sea la cantidad de bytes menor es la latencia del envío del datagrama.

Existen diversos problemas en la API que se documentan más adelante. Esta API es la única opción que existe en un navegador actual de establecer una conexión P2P sin hacer uso de complementos. Usar complementos daría la opción de usar una alternativa a WebRTC para establecer conexiones P2P entre navegadores, pero hay en marcha una hoja de ruta para bloquearlos los complementos lentamente a lo largo del tiempo con la intención de mejorar la seguridad, estabilidad y compatibilidad de la web entre navegadores, y en su lugar han expandido las API existentes bajo el nombre HTML 5.

6.8. STUN

STUN (15) es un protocolo que resuelve la mayoría de los problemas existentes a la hora de realizar una conexión P2P entre direcciones que se encuentran tras una NAT. NAT permite el uso compartido de una dirección IP por varias IP creando una subred. Todas las peticiones que se originan en la subred y están dirigidas fuera de la misma son multiplexadas para usar la misma IP, pero distintos puertos. NAT se encarga de asignar los puertos multiplexados y hace un seguimiento de estos para conocer, cuando llega un paquete, a que IP de la subred debe de ser enrutado.

Hay muchos tipos de NAT y no todas las NAT están implementadas siguiendo un estándar, teniendo como efecto problemas varios como el cambio de puerto asignado a una IP de la subred de forma aleatoria durante la transmisión de paquetes por UDP.

Debido al funcionamiento de NAT, un paquete entrante dirigido a un puerto que NAT no ha asignado con anterioridad a alguna IP de la subred no tiene a donde ir y por tanto es desechado, haciendo imposible la tarea de iniciar un servidor que espere conexiones entrantes. La solución para esta situación es configurar NAT para que redireccione el tráfico dirigido a cierto puerto a una IP y puerto concretos de la subred, pero esta configuración no siempre puede realizarse. Por ejemplo, espacios públicos o universidades no permiten definir redirecciones de puertos. STUN resuelve el problema mediante la técnica llamada *NAT hole punching* que consiste en:

1. Ambos futuros integrantes de la conexión contactan con un tercero (no es necesario que ambos contacten el mismo) que responde enviando la IP y el puerto del paquete recibido.
2. Un servicio de mensajería entre ambos futuros integrantes es usado para intercambiar esta información.
3. Ambos futuros integrantes envían paquetes a la IP y puerto indicado por el otro.
4. Si todo sale bien, los paquetes de ambos integrantes ahora son capaces de alcanzar al otro.

Para funcionar es necesario que la NAT no use distintos puertos dependiendo de la IP de destino, entre otros problemas. La mayoría de NAT funcionan con esta técnica.

STUN contiene las herramientas necesarias para confirmar que los integrantes de una conexión están estableciendo una conexión entre ellos y no con otros mediante el uso de una función hash que usa una clave intercambiada mediante el servicio de mensajería anterior, requiriendo por tanto que el servicio de mensajería sea privado y seguro.

El protocolo cuenta con una extensión llamada TURN que tiene el objetivo de redirigir paquetes, actuando como *relay*, para las situaciones en las que no haya éxito con la anterior técnica.

6.9. DTLS

DTLS (16) (17) es un protocolo que implementa encriptación para protocolos que envían datagramas, como UDP. Está basado en el protocolo más conocido TLS y provee de las mismas garantías. El protocolo no es muy popular por lo que no hay muchas implementaciones de este a diferencia de TLS. La similitud con TLS es intencional, con el objetivo de ser igual de seguro y evitar reinventar la rueda en asuntos de seguridad, consiguiendo por tanto que, si TLS es seguro, DTLS lo también lo sea.

Existen dos versiones, 1.0 y 1.2. WebRTC recomienda el uso de la versión 1.2 pero no todas las librerías criptográficas ofrecen ambas versiones. OpenSLL[35], la librería criptográfica más conocida, soporta ambas versiones, mientras que BouncyCastle[36], la única librería en C# que funciona con Unity, únicamente soporta la versión 1.0 a pesar de que afirman soportar la versión 1.2.

6.10. SCTP

SCTP (18) (19) es un protocolo perteneciente al nivel de transporte del modelo OSI. Es posible implementarlo tanto sobre TCP como sobre UDP y está diseñado con la intención de poder ser incorporado en una aplicación existente sin necesidad de modificarla, introduciendo SCTP como una capa intermediaria entre TCP/UDP y la aplicación. Ofrece transmitir paquetes sin duplicados, fragmentar paquetes grandes, entregar secuencialmente los paquetes con la opción

[35] https://www.openssl.org/
[36] http://www.bouncycastle.org/csharp/

de indicar el orden deseado de la recepción de paquetes de forma individual para cada mensaje, control de congestión, envío y recepción de latidos para comprobar si una dirección sigue siendo alcanzable, seguridad ante diversos tipos de ataques, enviar varios mensajes en un mismo paquete y abstraer la conexión permitiendo que múltiples direcciones de la capa de transporte inferior apunten a la misma dirección SCTP, haciendo el protocolo más resiliente a la pérdida de una conexión, entre otros.

La implementación del protocolo en los navegadores Mozilla Firefox y Google Chrome presenta problemas que se desarrollarán más adelante.

6.11. Datachannel

Datachannel (20) (21) es un protocolo diseñado para ser usado sobre SCTP para añadir a WebRTC la funcionalidad de enviar y recibir mensajes binarios y de texto UTF-8 (el protocolo hace distinción entre ambos tipos).

Este protocolo es relativamente simple ya que hace de intermediario entre SCTP y la aplicación, dando la opción de enviar texto que después convierte a formato binario y dando la opción de enviar datos binarios directamente. Dado que el envío y recepción de datos binarios ya está contemplado en SCTP, Datachannel hace uso de ello haciendo que el protocolo carezca de complejidad.

6.12. WebSocket

WebSocket (22) es un protocolo asentado sobre el protocolo HTTP que permite la interacción entre un servidor y un cliente de modo que ambos pueden enviar y recibir datos en cualquier momento mientras se mantenga la conexión, con la intención de permitir el intercambio de datos en tiempo real. Está soportado por la mayoría de los navegadores web, incluyendo todos los navegadores web más importantes. Se implementó por primera vez en Google Chrome en 2009 y desde entonces ha ganado popularidad. Navegadores web que no soportan WebRTC es muy probable que sí soporten WebSocket.

Hasta la aparición de este protocolo la comunicación en tiempo real entre cliente y servidor únicamente era posible mediante el uso de complementos en el navegador. Hasta la aparición de WebRTC, WebSocket era la única opción viable de realizar un videojuego en línea para navegador web sin el uso de complementos. El protocolo es relativamente simple comparado con WebRTC, siendo por tanto la opción preferida para este tipo de aplicaciones por los desarrolladores incluso hoy en día a pesar de los inconvenientes de TCP.

Unity implementa este protocolo para la comunicación en línea entre WebGL y otras plataformas, siendo además la única opción disponible para realizar un juego en línea con Unity para WebGL usando las herramientas que Unity incluye por defecto.

Este protocolo no permite la comunicación P2P a diferencia de WebRTC ya que requiere que uno de los participantes en la conexión actúe como servidor, pero los navegadores web no ofrecen la opción de actuar como servidor. Además, WebSocket usa TCP, siendo menos preferible a UDP y por tanto a WebRTC para aplicaciones como videojuegos en línea, como se explicó anteriormente.

6.13. Wireshark

Wireshark[37] es una aplicación gratuita de código abierto con licencia GNU GPL que es capaz de capturar todo el tráfico que pasa por una interfaz de red para analizarlo posteriormente. Permite filtrar paquetes, diseccionarlos dependiendo del protocolo usado en ellos, conocer cuándo fueron capturados y en qué orden, puertos y direcciones IP del emisor y receptor, contenido del paquete y protocolos usados. Ofrece la opción de guardar los paquetes capturados para poder inspeccionarlos con posterioridad.

Esta herramienta ha sido de gran utilidad en el proyecto durante la implementación de WebRTC al permitir analizar tráfico WebRTC intercambiado por navegadores web y poder compararlos después con el tráfico generado por la aplicación, pudiendo así encontrar diferencias entre ambos para localizar problemas y poder arreglarlos.

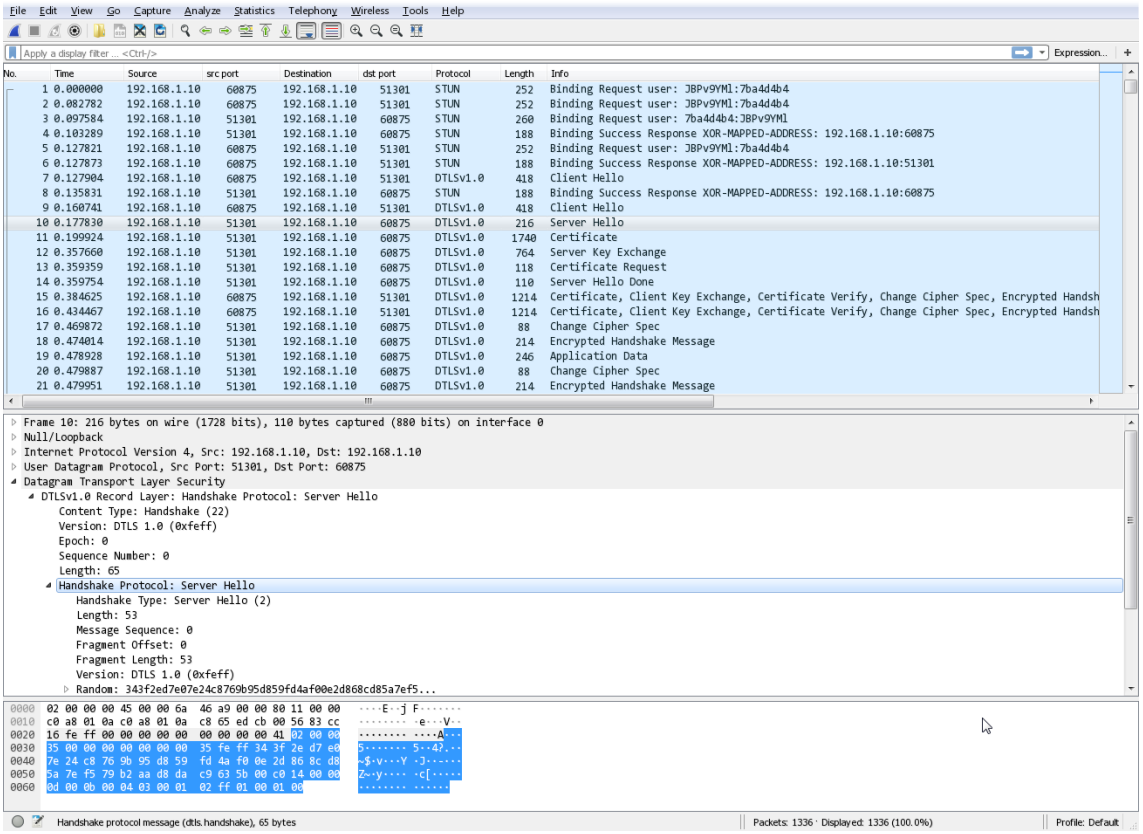

Figura 10 Captura de pantalla de la aplicación Wireshark visualizando una captura.

6.14. Clumsy

Clumsy[38] es una aplicación gratuita de código abierto con licencia MIT que permite simular condiciones adversas en la red. Permite añadir latencia, perder paquetes, retrasar paquetes en ráfagas de tiempo, duplicar paquetes, desordenar paquetes y alterar paquetes con información

[37] https://www.wireshark.org/
[38] https://github.com/jagt/clumsy

aleatoria. Aunque no es capaz de simular todas las posibles situaciones problemáticas que se pueden dar en la red, sí es capaz de simular una gran cantidad de ellas, haciendo que este herramienta sea de utilidad para comprobar el funcionamiento del complemento en multitud de escenarios. Aunque es posible implementar esta herramienta como parte del complemento, la existencia de esta facilita la realización de pruebas.

La aplicación además permite filtrar qué paquetes afectar usando reglas del filtro, pudiendo filtrar por IP, puerto, protocolo y dirección de los paquetes, tanto entrantes como salientes.

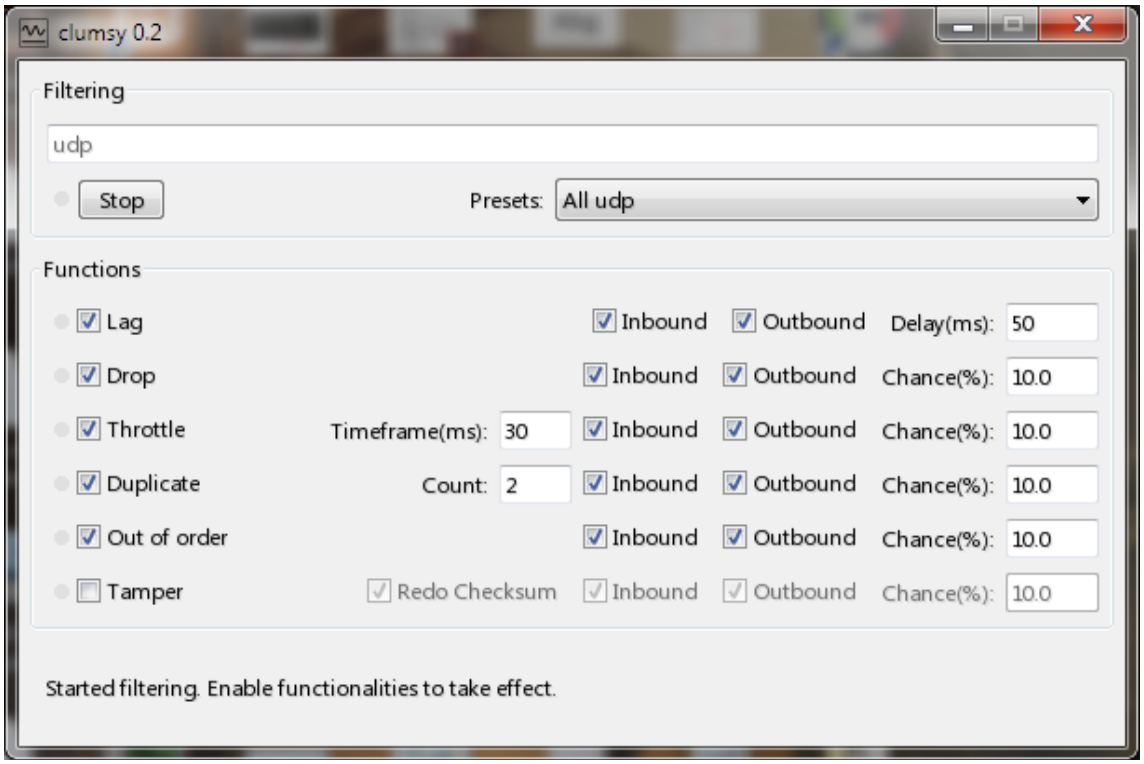

Figura 11 Captura de pantalla de la aplicación Clumsy.

7. Desarrollo de la solución propuesta

7.1. Mapa de características

Hay una gran cantidad de características deseables para este complemento, pero no todas ellas pueden ser llevadas a cabo en el tiempo limitado disponible. Se ha recopilado un listado de todas las características deseables y se han ordenado por prioridad. Para facilitar la lectura se ha incluido separadores para indicar qué características se han incluido en cada MVP, además de incluir esta información en el punto correspondiente a cada MVP. Como se puede observar más adelante, no todas las características deseadas han sido llevadas a cabo, quedando pendientes para ser realizadas más adelante. El listado se encuentra a continuación:

1. **Conexión directa usando sockets UDP**
   Establecimiento de una conexión usando el protocolo UDP usando la API `socket` de C#, que no está orientado a conexión.

2. ***NAT hole punching***
   Implementar STUN con la conexión directa para que funciona con un servidor de señales al igual que WebRTC.

3. **WebRTC**
   Establecer una conexión usando WebRTC con, por lo menos, un navegador web.

4. **Conexión y desconexión de parejas**
   Lógica básica de conexión y desconexión, introducción del administrador de conexiones `NetManager` basado en la librería LiteNetLib.

5. **Servidor maestro**
   Servidor intermediario para conexiones directas usando *NAT hole punching* y WebRTC.

PRIMER MVP

6. **Protocolo y capa de mensajería**
   Implementación de la segunda capa que trabaje con una interfaz para mensajería.

7. **Compresión diferencial**
   Optimización de la capa de mensajería explotando el protocolo para implementar compresión diferencial en los mensajeros.

8. **Mensajeros con distintas calidades de servicio**
   Implementación de los mensajeros básicos, tanto para enviar búfers como para enviar paquetes.

9. **Prioridad de mensajeros**
   Los mensajeros pueden tener prioridad de envío en caso de no poder transmitir todos los que lo necesitan.

10. **Protocolo capa Unity**
Protocolo para Unity sobre los mensajeros de la capa de mensajería. Identificador entre actores en línea, instanciación y destrucción de actores, carga y descarga de escenas, interfaz para establecer y terminar conexiones y más.

11. **Sincronizador de `Transform`**
Script sincronizador de posiciones, rotaciones y escalas de actores.

12. **Sincronizador de `Animator`**
Script sincronizador de animaciones en Unity usando una máquina de estados.

13. **RPC**
Llamadas remotas a funciones en otros actores.

14. **Gráfica de uso de la red con desglose de los paquetes enviados y recibidos**
Gráfica en tiempo real de los paquetes entrantes y salientes junto con el contenido de dichos paquetes y bits usados para cada cosa.

15. **Indicar preferencia entre rendimiento de CPU o compresión de ancho de banda**
Permitir elegir entre usar escrituras y lecturas en búfer desalineadas a nivel de bits, ahorrando ancho de banda, pero aumentando el uso de CPU, o alineadas a nivel de byte, gastando más ancho de banda, pero usando menos CPU.

SEGUNDO MVP

---

16. **Límite de ancho de banda y control de congestión**
Funcionar correctamente en situaciones de red adversas en caso de congestión en la red, además de permitir limitar el ancho de banda usado.

17. **Conexión con contraseña**
Permitir crear partidas privadas protegidas por una contraseña u algún otro método de autenticación que limite qué jugadores están autorizados para participar en la partida.

18. ***LAN discovery***
Buscar servidores en la red local sin necesidad del servidor maestro, para juego en red sin necesidad de internet mientras que los jugadores estén en una red local.

19. **Servidor autoritativo**
Modelo de sincronización de servidor autoritativo para prevenir trampas. Modelo más común en los juegos de acción rápidos.

20. **Captura de tráfico de red y retransmisión de este para grabar partidas, permitiendo su posterior visualización**
Permitir grabar partidas para revivirlas en otro momento mediante el uso de un cliente falso, además de retransmitir la partida grabada mediante un servidor falso.

21. **Más métodos de sincronización de estado del juego, como usando las físicas**
Más métodos de sincronización con los que se puede implementar modelos distinto de sincronización del juego.

22. **Integración con Steamworks**
Implementar Steamworks en la librería en la capa de conexión.

23. **Compresión Huffman**
Compresión del búfer de salida (y descompresión del de entrada) mediante el algoritmo Huffman, configurable usando grabaciones previas de búfers enviados y recibidos.

24. ***Host migration***
Permitir que en caso de que un jugador que actúa como servidor cierre el juego, el resto de los jugadores pueda continuar jugando en línea gracias a seleccionar y conectarse a uno de los jugadores restantes que hará de nuevo servidor.

25. **Servidores TURN**
Soporte para servidores TURN que hacen de intermediarios entre jugadores que no pueden establecer una conexión directa entre ellos y son compatibles con WebRTC.

26. **Limitar memoria usada**
Optimización del uso de memoria limitando colecciones de tamaño arbitrario para evitar desbordamientos de memoria.

27. **TCP**
Implementación del protocolo TCP aparte de UDP para conexión directa y WebRTC.

7.2. Desarrollo del primer MVP

El primer MVP consta de las características 1 a la 5, ambas incluidas.

Durante el desarrollo, con el objetivo de evitar reinventar la rueda y optimizar el tiempo, se ha intentado usar todas las librerías existentes posibles, siempre prestando atención a las licencias de las mismas.

Al inicio del proceso se intentó usar la librería HumbleNET[39] [40], que ofrece WebGL para C# y Unity bajo una licencia permisiva. Consta de una librería programada en C/C++, que contiene la lógica, y una envoltura en C# para dicha librería que además funciona en Unity. Tristemente no cumple con su promesa ya que no sólo resulta no funcionar en Unity, sino que además no es mantenida con regularidad. Dado que la librería no está completa y requiere conocer C++ para completarla y arreglarla, para poder usar esta librería sería necesaria una inversión de tiempo para arreglarla y además aprender C++. Esta librería permaneció como una opción en caso de no encontrar otra solución viable, siendo por ello finalmente descartada.

A continuación se encontró la librería FM.IceLink, que ofrece una implementación de WebRTC completamente en C# y además compatible con Unity, pero a un precio económico lo suficientemente elevado como para no elegirla.

[39] https://github.com/HumbleNet/HumbleNet
[40] https://hacks.mozilla.org/2017/06/introducing-humblenet-a-cross-platform-networking-library-that-works-in-the-browser/

A partir de este punto se decide implementar WebRTC en C#. Para implementar WebRTC es necesario contar con una implementaciones de ICE, STUN, DTLS, SCTP y Datachannel.

Dado que se va a implementar en C# es necesario usar la API `Socket` de C#. Esta API crea presión en el recolector de basura, que Unity penaliza resultando en pausas durante el juego cuya frecuencia y duración aumenta a mayor sea la presión sobre el recolector de basura. Para solventar este problema con la API se ha creado una librería llamada No-gc-sockets[41] que reduce la cantidad de bytes que el recolector de basura tiene que reclamar por las dos funciones que se usan en este complemento. Estas dos funciones son `SendTo`, que de reclamar 80 Bytes ahora reclama 0 Bytes, y de la función `ReceiveFrom`, que de reclamar 250 Bytes ahora reclama 82 Bytes. Esto se consigue mediante la extensión de la clase `IPEndPoint` con una implementación incompleta que no realiza copias de objetos a diferencia de la implementación original. Los 82 Bytes reclamados por `ReceiveFrom` se deben a la creación de un objeto por el CLR de C# y por tanto no se puede evitar a no ser que se use otra función. Para evitar problemas, esta librería mantiene los objetos ocultos al exterior y expone únicamente lo necesario para usar la librería: Una clase estática con las funciones `SendTo` y `ReceiveFrom` que funcionen de forma equivalente a las de la API `Socket`.

Para ICE se ha usado ICE4J[42] y SipSorcery[43]. La implementación de ICE por SipSorcery es mucho más completa y compleja de lo necesario por lo que se implementó una nueva versión más simple, dejando de lado la implementación de SipSorcery.

SipSorcery cuenta con una implementación de STUN, pero contiene dos problemas:

- Crea y destruye objetos continuamente, concretamente múltiples pequeños array de bytes que se mantienen en memoria hasta que son copiados en el búfer (otro array de bytes) que será enviado. Ocurre el mismo problema a la hora de recibir un mensaje STUN. Esta creación y destrucción de objetos continua genera una carga sobre el recolector de basura que, como mencionado anteriormente, Unity penaliza.
- Realiza envíos de ping a pesar de existir tráfico en la conexión. Aunque este es un problema leve, la generación de tráfico innecesaria en un complemento como este debe de ser minimizada en la medida de lo posible.

A pesar de que la implementación de STUN de SipSorcery es funcional, debido a estos problemas se implementó una nueva librería de STUN[44] a partir de una librería existente en Java[45] con un diseño sencillo. Esta nueva librería está diseñada para no crear y destruir objetos. Junto a esta nueva librería se implementó otra librería llamada BBuffer[46] que contiene una estructura C# llamada `ByteBuffer`, inspirada en la API de Java `ByteBuffer`[47], que envuelve una sección de un array de bytes y lo provee de funciones para leer y escribir tipos simples con rapidez, tanto en Big Endian como en Little Endian.

[41] https://github.com/forestrf/No-gc-sockets
[42] https://github.com/jitsi/ice4j
[43] https://github.com/sipsorcery/sipsorcery
[44] https://github.com/forestrf/stun
[45] https://github.com/hf/stun
[46] https://github.com/forestrf/BBuffer
[47] https://docs.oracle.com/javase/7/docs/api/java/nio/ByteBuffer.html

BBuffer cuenta con una estructura C# similar a `ByteBuffer`, llamada `BitBuffer`, que permite en su lugar escribir cantidades de bits arbitrarias, no necesariamente 8, para optimizar el ancho de banda. Cuando una lectura o escritura se produce en un índice cuyo múltiplo es distinto a 8 (alineado a un byte) es necesario realizar cálculos extra en la CPU y por tanto resulta más lento, por lo que más adelante se da la opción en la capa de mensajería de alinear el búfer al siguiente byte antes de escribir cada mensajero.

Para DTLS se ha usado BouncyCastle. La librería es lenta y crea y destruye objetos con regularidad, pero debido a la complejidad de la misma es complicado de solucionar. Además, la librería presenta dos problemas:

- No implementa DTLS 1.2 correctamente. Por ello no es posible realizar una conexión con un navegador sin antes modificar la petición ICE manualmente para que BouncyCastle actúe siempre como servidor y pueda así forzar el uso de DTLS 1.0, que sí funciona, al navegador web. No realizar esto conseguiría que el navegador web fuerce el uso de DTLS 1.2 y la conexión no consiga establecerse.
- Todo envío redirigido a BouncyCastle para que lo encripte y sea enviado por el `socket` genera dos paquetes en lugar de uno. Este problema duplica la cantidad de paquetes enviados sin necesidad, suponiendo un problema de rendimiento que ha limitado la cantidad de envíos por segundo en Unity en 30 cuando puede enviar 60 paquetes por segundo sin problema usando una conexión directa sin WebRTC.

Estos problemas son graves, pero no tienen solución sencilla sin implementar DTLS de nuevo. Esta tarea no sería sencilla al tratarse de un protocolo usado para criptografía y una implementación del mismo puede invalidar la seguridad que el protocolo ofrece. Por falta de recursos se ha decidido dejar y solucionar en el futuro.

Para SCTP se ha portado la librería SCTP4J[48] de Java a C# 3.5, cambiándole el nombre a SCTP4CS[49]. Gracias a este proceso se ganó conocimiento suficiente para implementar de nuevo SCTP, esta vez siguiendo los RFC 3758 y 4960, aunque no por completo, para optimizar la librería que debido al diseño de su implementación carecía del rendimiento necesario. SCTP requiere funcionalidades que no han sido implementadas porque no son usadas por WebRTC. Una funcionalidad que es usada por WebRTC pero se ha ignorado en este complemento es la posibilidad de enviar mensajes ordenados y con sin pérdidas. Esta decisión se ha tomado porque se pretende implementar esta lógica en la capa superior, de mensajería. Esta decisión ha supuesto perder varios bytes que SCTP siempre usa para poder mantener control sobre las calidades de servicio. En el futuro la capa de mensajería podría detectar si la capa de transporte tiene la calidad de servicio ordenada y sin pérdidas, para hacer uso de ella si fuera posible, pero esta decisión aumentaría la complejidad del diseño y no se ha llevado a cabo.

A partir de este punto WebRTC ya funciona y es posible establecer una conexión con un navegador web. El desarrollo ha sido llevado a cabo como una librería en Visual Studio, así que se copia la librería a Unity y se escribe una librería en JavaScript para que WebRTC funcione en

[48] https://github.com/pipe/sctp4j
[49] https://github.com/forestrf/sctp4cs

el navegador, usando como referencia una librería llamada Unity-Netcode.IO[50] que contiene una librería JavaScript similar.

Para la creación de la API se ha usado la librería LiteNetLib como inspiración por la sencillez de la misma.

Para terminar, para el servidor maestro o servidor de señales se intentó usar opciones existentes, entre ellas el servidor maestro que ofrece Unity, pero fue desechado porque no está diseñado para las necesidades de WebRTC, que requiere de un servicio de mensajería entre los jugadores que quieren establecer una conexión. Usando Websocket-sharp[51] se implementó un servidor de señales simple que ha resultado suficiente para este proyecto. La implementación del servidor maestro está separada de la capa de conexión, haciendo posible usar uno distinto.

7.2.1. Experimento 1

El primer experimento consistió en permitir que una persona cercana pudiera usar la librería para después obtener una opinión de ella. Se documentó la API expuesta por la librería y se cedió al participante la documentación junto con el código fuente comentado de la librería. El tiempo límite se estableció en 15 días.

El resultado del experimento fue que no había suficiente funcionalidad en el complemento. El complemento únicamente ofrecía la habilidad de realizar conexiones entre jugadores en cualquier plataforma de Unity junto con un servidor maestro que facilitaba la creación de partidas. El participante indicó que es atractivo poder usar WebGL, pero que la complejidad necesaria para hacer algo con la librería era demasiado grande.

De este experimento se pudo extraer las siguientes conclusiones:

- Era necesario tener al menos la capa de mensajería para ofrecer distintas calidades de envío comparables a otras librerías como Lidgren o LiteNetLib en lugar de únicamente UDP.
- El objetivo del complemento es poder realizar videojuegos en línea. Con esa promesa el participante aceptó probar el complemento, pero este estaba lejos de cumplir dicha promesa.
- El camino que se estaba siguiendo con el complemento no fue debatido por lo que se pudo seguir en esa dirección.

7.3. Desarrollo del segundo MVP

El segundo MVP consta de las características 6 a la 15, ambas incluidas.

El desarrollo contempla las capas de mensajería y Unity, aunque también se alteró código en la capa de transporte para realizar mejoras y arreglar fallos.

[50] https://github.com/KillaMaaki/Unity-Netcode.IO
[51] https://github.com/sta/websocket-sharp

Primero se implementó el protocolo de comunicación sobre el que se asientan los mensajeros. Se implementó un sistema de notificación de paquetes recibidos en el protocolo, añadiendo un identificador único a cada paquete enviado para evitar paquetes duplicados y conocer el orden de envío de los paquetes, haciendo pública esta información para los mensajeros. Usando el identificador y el sistema de notificaciones se previene el procesado de paquetes duplicados y se puede conocer el orden de envío de los paquetes, soportando que estos lleguen desordenados. El protocolo integra también un sincronizador del reloj entre los dos integrantes de una conexión que permite a ambos conocer el estado actual del reloj del otro.

El algoritmo usado para la sincronización del tiempo es una variante de SNTP (23). Los paquetes enviados tienen una marca de tiempo usada por el protocolo para determinar el reloj remoto. La diferencia frente a SNTP es la discriminación de paquetes cuyo RTT tenga una desviación estándar superior a uno comparado con el RTT medio de las muestras capturadas.

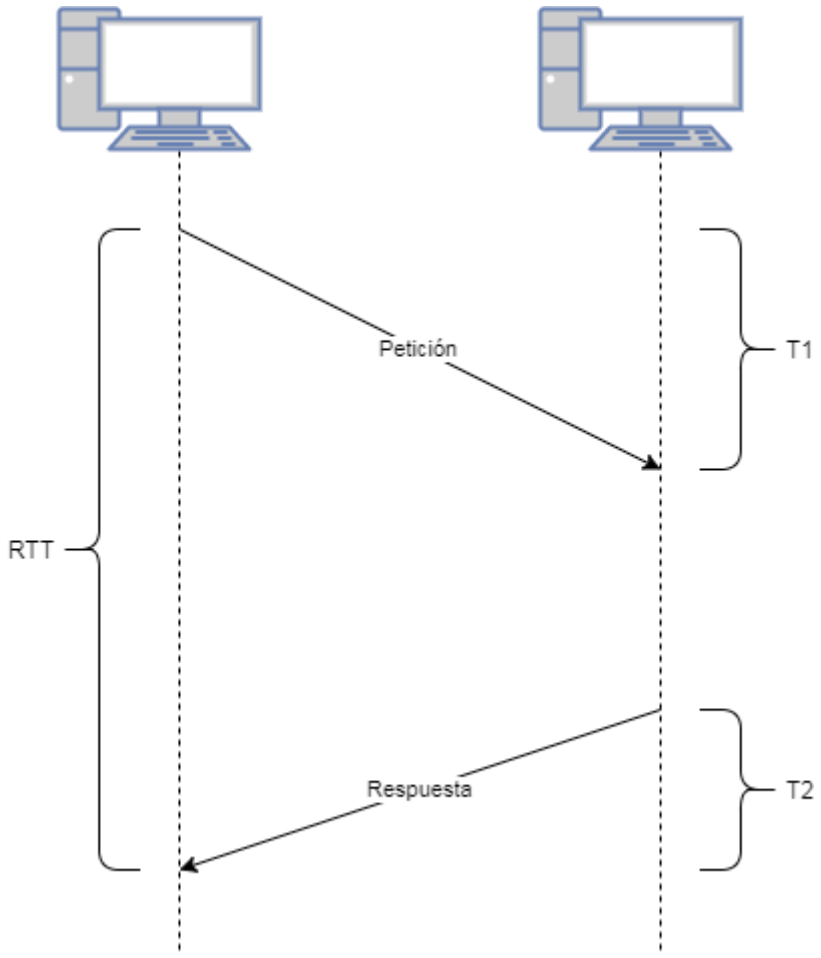

Figura 12 Diagrama temporal que muestra los tiempos en el envío y recepción de un paquete.

Usando la figura 12 como referencia, El error máximo del reloj sincronizado es *RTT* (*Round Trip Time*, tiempo de ida y vuelta de un paquete) y el error mínimo es el valor más grande entre $|RTT/2 - T1|$ y $|RTT/2 - T2|$, siendo un error cercano al mínimo posible el más probable en condiciones ideales. Tras múltiples mediciones en condiciones adversas de red y con deriva entre los relojes a sincronizar, el error converge hacia error mínimo. Las figuras 13 y 14 muestran la diferencia entre dos relojes sincronizados en distintas situaciones de red:

- La primera figura representa una situación idóneas, con un *RTT* aleatorio entre 6 ms y 20 ms, diferencia entre |*RTT/2 – T1*| y |*RTT/2 – T2*| máxima de 10%, 1% de paquetes perdidos, 5 % de paquetes retrasados, 30±5 envíos por segundo y deriva del reloj de 1,0000001.
- La segunda figura representa una situación adversa, con un RTT aleatorio entre 200 ms y 600 ms, diferencia entre |*RTT/2 – T1*| y |*RTT/2 – T2*| máxima de 30 %, 30 % de paquetes perdidos, 5 % de paquetes retrasados, 30±15 envíos por segundo y deriva del reloj de 1,000001.

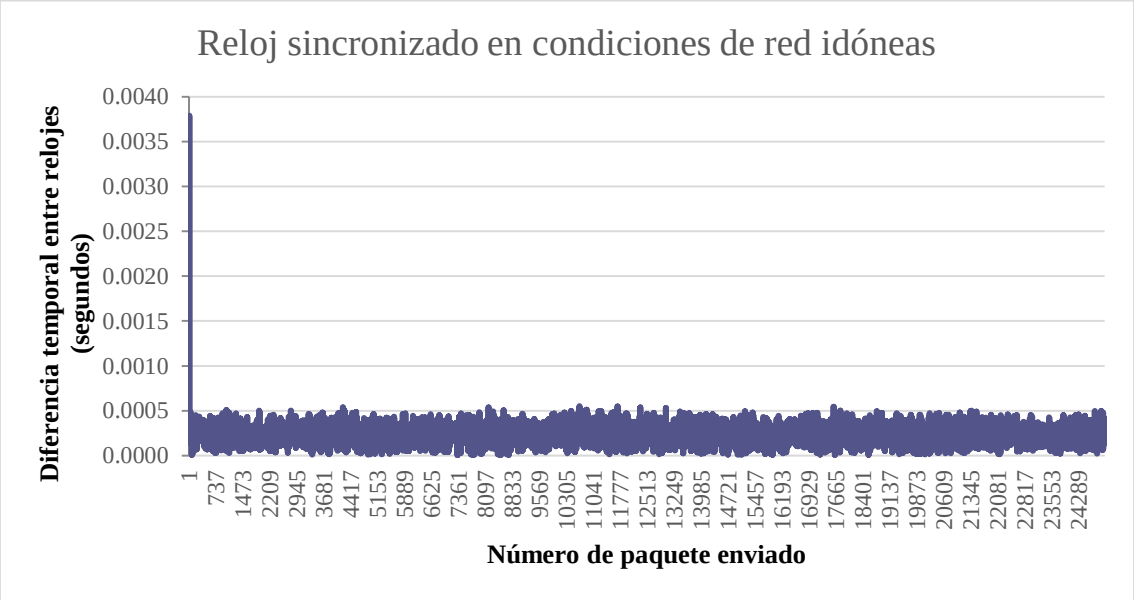

Figura 13 Gráfica que muestra la diferencia temporal entre dos relojes sincronizados en condiciones de red idóneas.

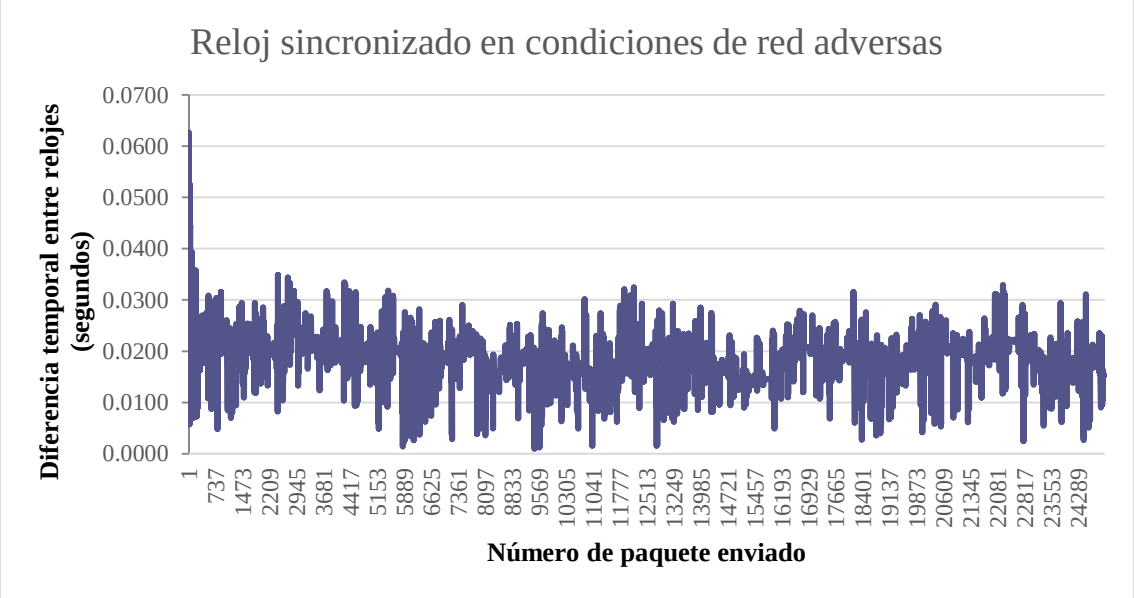

Figura 14 Gráfica que muestra la diferencia temporal entre dos relojes sincronizados en condiciones de red adversas.

Más adelante se implementó los mensajeros mediante una interfaz común a todos ellos. Los mensajeros se suscriben al protocolo, que cuando llega el momento de enviar comprueba qué

mensajeros necesitan escribir en el paquete que se enviará, se ordenan por prioridad y se les da acceso al búfer para que escriban en él. El mensajero será llamado más tarde para informarle de si el paquete alcanzó el destino o si muy probablemente no lo hizo. Debido al diseño del protocolo sólo se puede garantizar la entrega de un paquete mientras que la falta de entrega se supone no entregado pasada una cantidad de tiempo determinada, aunque puede que sí sea entregado pasada dicha cantidad de tiempo.

Debido a decisiones tomadas durante la implementación, la medición de *RTT* tiene una resolución menor que el tiempo transcurrido entre el dibujado de dos cuadros consecutivos del juego, ya que es durante el dibujado cuando se procesa la lógica del juego, incluido los eventos de red y, por tanto, se responde a un paquete entrante que ha llegado en cualquier momento entre el anterior cuadro y el actual. Si el juego dibuja un nuevo cuadro 60 veces por segundo, un valor común en los videojuegos debido a ser el número de veces por segundo más común que una pantalla puede actualizar la imagen que muestra por segundo, eso supone hasta 16 ms de error añadido en la medición.

Los primeros mensajeros que se implementaron permiten serializar estructuras de C#. Más adelante se encontró que estos mensajeros eran insuficientes para situaciones en las que se necesita enviar información de tamaño arbitrario, añadiendo por tanto mensajeros para búfers de tamaño arbitrario. Si el mensajero no puede escribir el búfer por completo en el espacio restante del paquete puede fragmentarlo y escribir sólo una parte de este, enviando el resto de búfer en paquetes sucesivos.

Una vez los mensajeros estuvieron listos, haciendo uso de los identificadores únicos se amplió el protocolo para permitir a los mensajeros realizar compresión diferencial. Para hacer uso de la compresión diferencial se añadió a la librería BBuffer funciones para escribir enteros con longitud variable, concretamente usando la compresión LEB128 (*Little Endian Base 128*) que permite usar menos bytes para representar números pequeños.

Para concluir la capa de mensajería se añade sobre el mensajero de estructuras enviadas de forma ordenada y sin pérdidas una clase llamada RPC, que facilita el uso del mensajero anterior para realizar llamadas remotas a métodos con parámetros (Si los parámetros no son de tipo numérico o cadena de texto es necesario crear una envoltura serializable para ellos con una estructura que implemente una interfaz concreta). Cuenta con la opción de restringir quién puede llamar de forma remota a un método local, indicando si se permite llamadas iniciadas en el servidor y/o en el cliente, pero esto no es una medida de seguridad ya que tanto un cliente como un servidor pueden intentar explotar vulnerabilidades de aquellos que están conectados a él.

Dada la capa de mensajería por finalizada, a partir de este punto se inicia el desarrollo de la capa de Unity, la parte con la que interactuarán en mayor medida los usuarios del complemento.

Primero se implementó el protocolo de sincronización global del juego. Esto es carga y descarga de niveles, instanciación y destrucción de objetos del juego sincronizados entre las instancias de Unity conectadas, administración de instancias de Unity conectadas entre sí y administración de los jugadores que se encuentran en una instancia de Unity, ya que es posible que múltiples jugadores usen la misma instancia de Unity para jugar. Este protocolo se ha implementado por completo sobre un mensajero de búfers de tamaño arbitrario, con calidad de servicio de tipo ordenado y sin pérdidas.

Antes de implementar los scripts de Unity para sincronizar el estado de los objetos se ha investigado sobre métodos de desarrollo y optimización de estos usando guías de la competencia (24) (25) entre otras fuentes mencionadas en el punto de introducción.

Para la sincronización del estado de los objetos del videojuego se ha decidido usar la técnica de interpolación de estados (5) por su relativa sencillez comparada con otras, aunque añade latencia. Usando esta técnica se ha creado dos scripts, uno para sincronizar el componente de Unity del tipo `Transform` y otro para sincronizar el componente de Unity de tipo `Animator`. Ambos han sido implementados usando uno o más mensajeros de estructuras de tipo ordenado con pérdidas.

Para completar la capa de Unity se implementó una gráfica que permite visualizar el tráfico en detalle. Esta gráfica está inspirada en el conjunto de herramientas de análisis del tráfico de red que se encuentran en el videojuego HALO:REACH (3). El gráfico desglosa los bits escritos en cada paquete para poder realizar análisis de estos y facilitar la realización de optimizaciones en el uso del ancho de banda. Las figuras 15 y 16 muestran la representación visual de las estadísticas de red. En ellas, el color amarillo representa cabeceras mientras que el color verde representa datos.

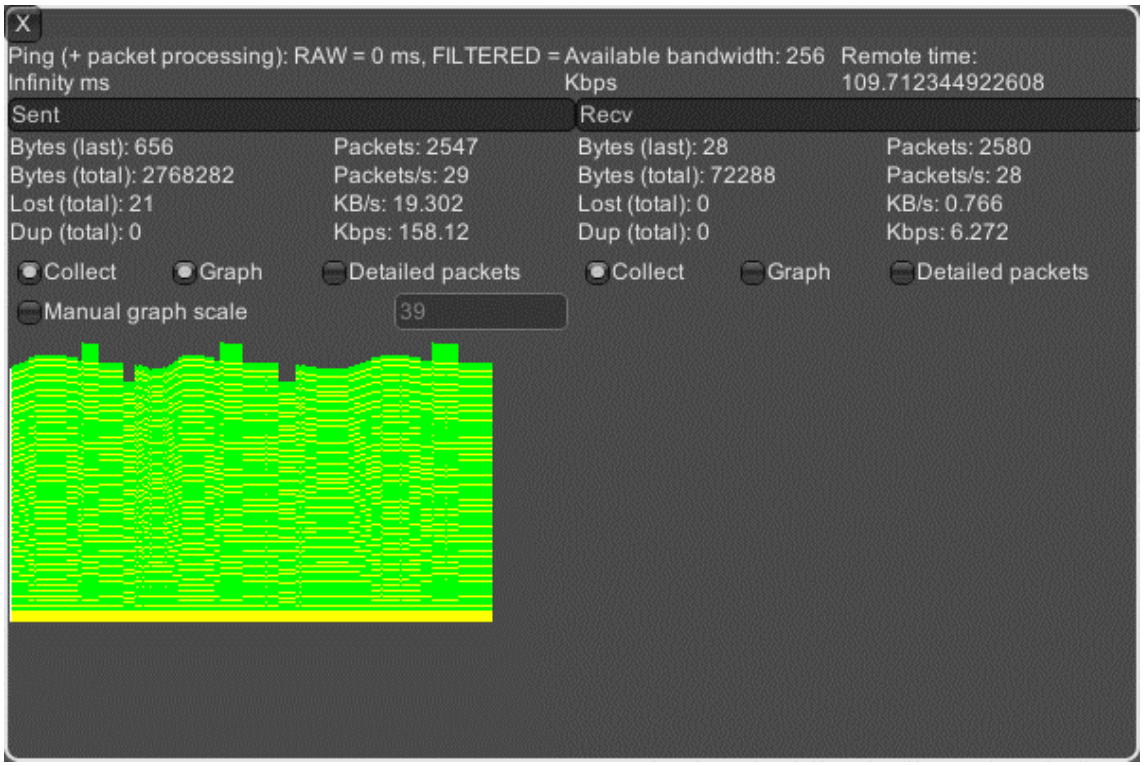

Figura 15 Visualización del gráfico del tráfico saliente en red.

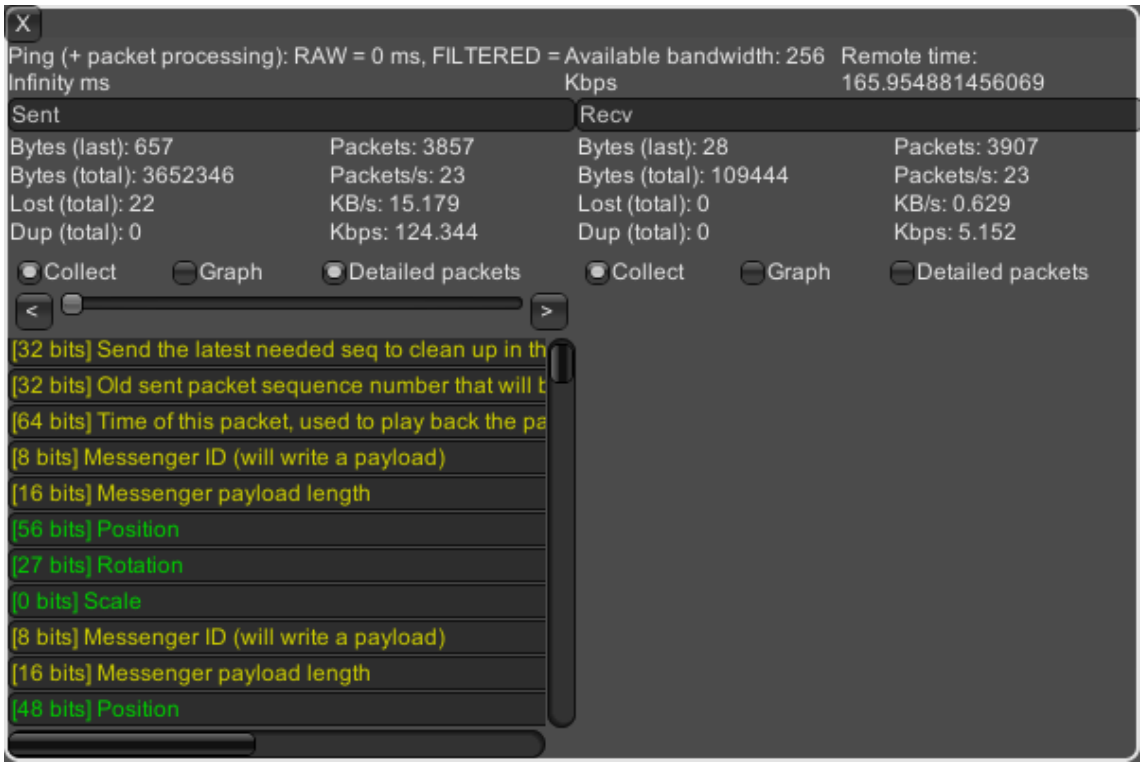

Figura 16 Visualización del desglose de un paquete del tráfico saliente en red.

7.3.1. Experimento 2

El segundo experimento consistió en permitir que 6 personas desconocidas, pertenecientes a un grupo aficionado al desarrolladores de videojuegos, probaran el complemento. Todos los participantes tenían interés en realizar un videojuego en línea y para ello estaban dispuestos a usar el complemento. Esta vez no se estableció una fecha límite y en su lugar se dio la indicación de que informaran sobre su opinión del complemento mientras lo usaban e informaran de problemas o dificultades que fueran surgiendo. Se les facilitó el código fuente comentado junto con documentación de la API y un manual de cómo empezar a usar la librería. Se les dio la opción de poder modificar el código del complemento y poder compartir sus cambios usando Git, pero ninguno de los participantes alteró el complemento. Debido a que ningún participante de la prueba dominaba español y que todos suelen usar inglés para relacionarse en la comunidad en línea de la que provienen, los manuales se encuentran redactados en inglés.

El resultado del experimento fue favorable. El complemento era sencillo de usar, pero la existencia de bugs, fallos menores de diseño y carencia de más opciones de configuración de la librería usando la API suministrada supusieron grandes problemas.

Uno de los participantes, que se encontraba desarrollando un juego similar a Minecraft y todavía no había implementado la funcionalidad de jugar en línea, consiguió que parte del juego funcionara en línea (permitiendo interactuar con otros jugadores y los bloques que conforman el terreno) en tan sólo 3 días y sin contar con experiencia previa en programación un videojuegos en línea.

Otro de los participantes requería de un modelo de sincronización distinto en la capa de Unity para poder realizar un videojuego de acción contra otros jugadores en línea. El diseño de la capa actual permite la inclusión de modelos distintos, pero debido a la complejidad de los modelos que requería y que el participante carecía de experiencia para llevar a cabo la tarea, decidió pedir que se incluyera el modelo en el complemento. Esto es similar a lo que Unity pretende realizar en el futuro con una nueva librería que reemplazará a UNet, que se basará en plantillas sobre las que desarrollar el juego y que cada una está basada en un modelo de sincronización distinto, viniendo cada una con los scripts más adecuados para dicho modelo y para el tipo de videojuego que se puede realizar con dicha plantilla.

Un participante que implementó su propio complemento en el pasado comprobó una notable reducción del uso de del ancho de banda entre este complemento y el suyo.

La opinión sobre la capa de mensajería y la lógica sobre la que trabaja fue bien recibida y fácil de comprender, aunque no así la API expuesta para programar mensajeros propios o hacer un uso correcto de los mensajeros ya creados (los mensajeros proveen de las calidades de servicio).

Por último, pidieron más funcionalidades no existentes: Permitir cambiar el propietario de un objeto en línea durante el juego y simplificar la API al igual que la lógica del complemento.

De este experimento se pudieron extraer las siguientes conclusiones:

- El complemento se podía usar para realizar un videojuego en línea, pero estaba lejos de estar listo para público en general al faltar funcionalidades que suelen estar presentes en otros complementos o que simplemente son convenientes, no siendo ideal comercializarlo todavía.
- La complejidad de la API era un problema y requería simplificación, sobre todo en la capa de mensajería.
- Debido a la complejidad de la API, implementar modelos de sincronización nuevos era complicado.
- El uso del ancho de banda era reducido comparado con otras alternativas.
- El camino seguido para desarrollar el complemento era bueno al no haber surgido ningún problema grave relacionado con las partes más importantes del complemento.

Cabe destacar que la decisión tomada de no dar una fecha límite para los participantes se debe a la disposición de los mismos por seguir trabajando como probadores en el futuro, pues tienen interés en el complemento al tener experiencia previa con otros complementos y ver una ventaja en usar este o bien prefieren usar un complemento en desarrollo cuyo creador está en contacto directo con ellos.

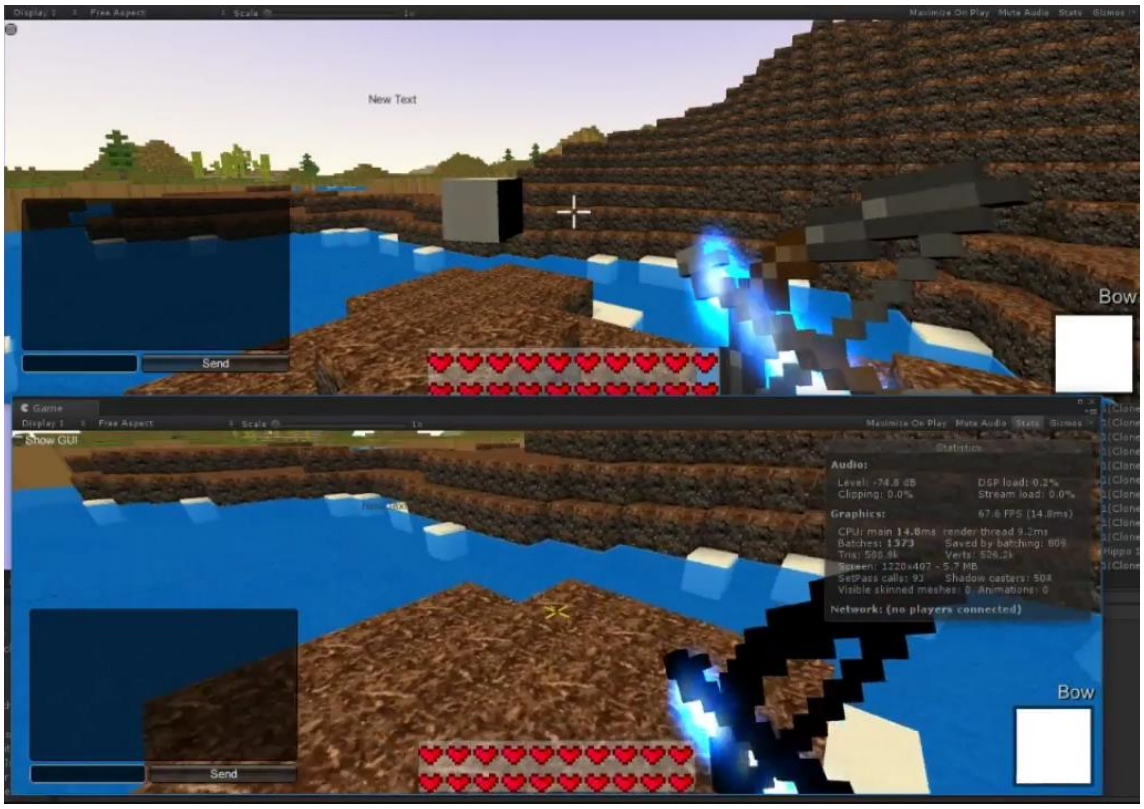

Figura 17 Captura de pantalla de una copia de Minecraft hecha en Unity por uno de los participantes del secundo MVP en la que se muestra a dos jugadores jugando juntos.

7.4. Métricas

- Número de archivos de código C#: 221
  - Normal: 186
  - Pruebas unitarios: 35
- Cantidad de pruebas unitarias: 135
- Cantidad de líneas de código: 19.146
  - Normal: 15.757
  - Pruebas unitarias: 3.389
- Cantidad de líneas de comentario: 2.143
  - Normal: 1.980
  - Pruebas unitarias: 163
- Cantidad de repositorios creados: 7
- Cantidad total de *commits* en GIT[52]: 842
  - Repositorio principal: 446
  - Repositorio para los participantes en los experimentos: 91
  - Repositorio BBuffer: 62
  - Repositorio HashUtils: 7
  - Repositorio No-gc-sockets: 18

[52] Contabilizado usando el comando GIT *git rev-list HEAD --count*

  - Repositorio Stun: 106
  - Repositorio sctp4cs: 112
- Número de participantes en los experimentos: 7
- Estadísticas en CPU FX 8350 y Windows 7: Tabla 2 y figuras 20, 21, 22 y 23.

	AshNet directo	AshNet WebRtc	LiteNetLib	Lidgren
Max. MB/s	35,61	14,44	2,99	16,77
Max. pps	36095	13686	3462	18175
CPU				
1 Cliente, 30 pps	0.08	0.29	0.06	0,3
10 Clientes, 30 pps	0.26	0.31	0.29	0,32
100 Clientes, 30 pps	1.5		1.47	0,9
500 Clientes, 30 pps	4.41		11.83	5,04
1000 Clientes, 30 pps	7.2			10,4
Memoria				
1 Cliente, 30 pps	22.5	26.2	21,79	21,67
10 Clientes, 30 pps	26.81	30.25	21,95	22,01
100 Clientes, 30 pps	74.6		24,31	22,55
500 Clientes, 30 pps	151.1		851,61	26,18
1000 Clientes, 30 pps	332.85			45,86

Tabla 2 Comparación de estadísticas entre AshNet y librerías similares.

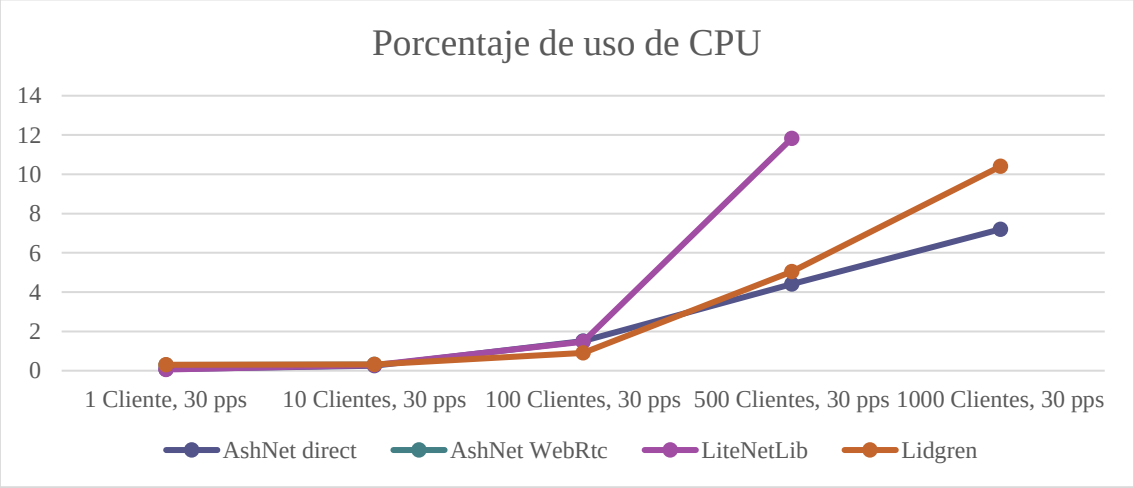

Figura 18 Porcentaje de uso de CPU de los datos representados en la tabla 2.

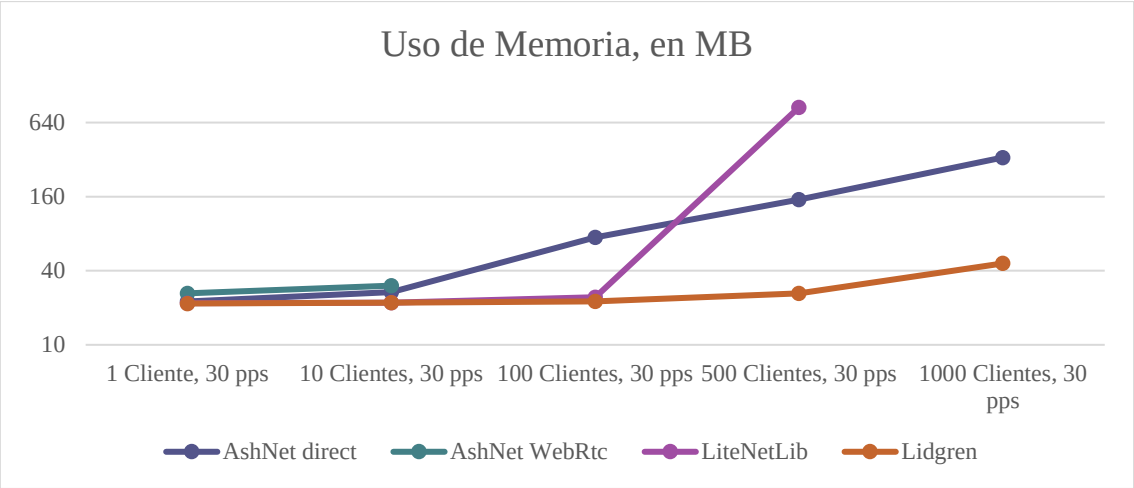

Figura 19 Uso de memoria, en megabytes, de los datos representados en la tabla 2.

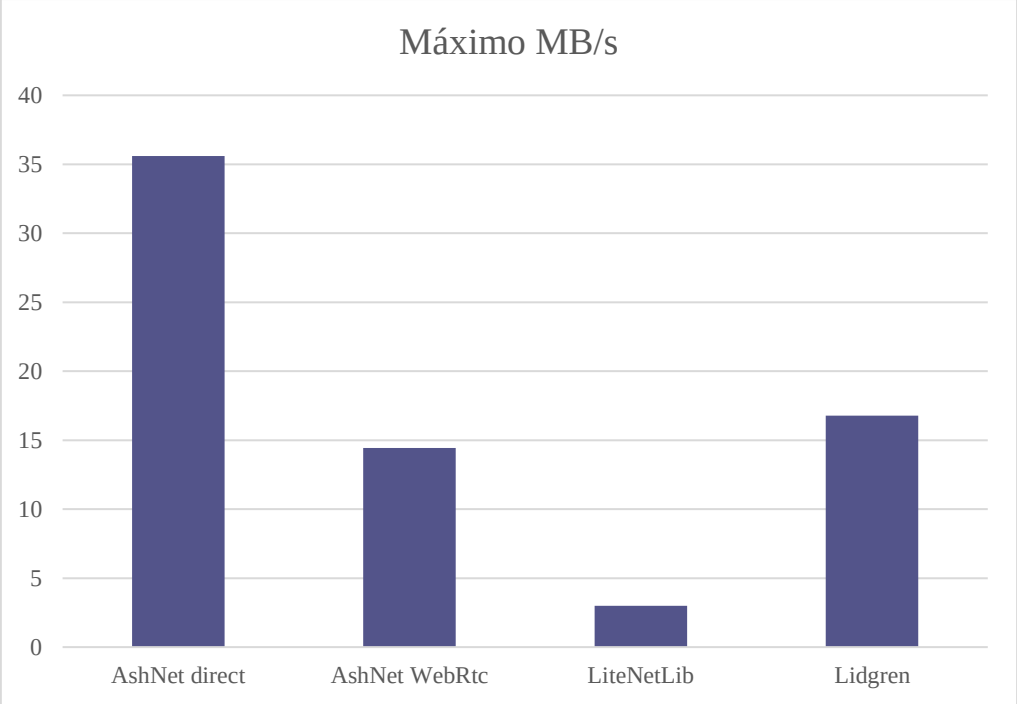

Figura 20 Cantidad máxima de megabytes por segundo de los datos representados en la tabla 2.

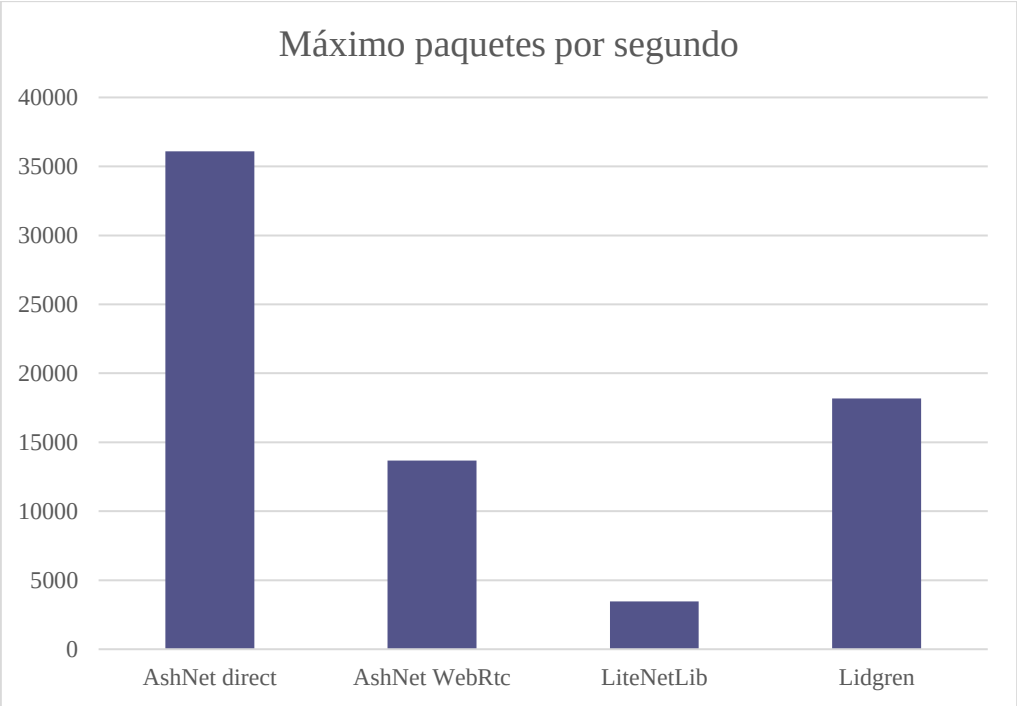

Figura 21 Cantidad máxima de paquetes por segundo de los datos representados en la tabla 2.

8. Implantación

El complemento está diseñado para que su implantación sea lo más sencilla posible. La implementación se divide en dos tareas, siendo la primera tarea optativa para hacer uso del complemento.

8.1. Despliegue del servidor maestro o servidor de señales

Primero es necesario disponer de un servidor capaz de ejecutar una aplicación desarrollada en C#. Tanto Windows, Max como Linux pueden realizar esta tarea. Si esto no es posible o deseable sería necesario implementar un servidor maestro en un lenguaje distinto. En este caso se usará un VPS con Linux. Los pasos para seguir se detallan a continuación:

1. Usar una herramienta como MobaXterm[53] para conectarse al VPS.
2. Instalar mono siguiendo las instrucciones de instalación detalladas en la página web oficial del proyecto mono[54].
3. Compilar el proyecto SignalingServer, que se encuentra en la solución C# AshNet.
4. Una vez compilado, enviar al VPS los archivos generados y dar permisos de ejecución al archivo SignalingServer.exe.
5. Crear y habilitar un servicio que ejecute el comando `mono SignalingServer.exe 8182 /nombreServicio` al arrancar el VPS. A partir de este momento el servidor maestro está ejecutándose y escuchando en el puerto 8182 bajo el servicio WebSocket `/nombreServicio`. Puerto y servicio son configurables. Es responsabilidad del encargado de desplegar la aplicación hacer que el servicio ejecute una nueva instancia del servidor en caso de que deje de se detenga el proceso.

El servidor maestro incluido en el proyecto es un ejemplo simple que no implementa protección frente a ataques externos ni está optimizado. Está diseñado para servir como guía de cómo tiene que ser un servidor maestro o como un servidor maestro temporal con el que hacer pruebas. Por tanto, no debe de ser desplegado en un entorno de producción.

8.2. Despliegue del complemento en Unity

Aunque el proceso de despliegue es sencillo, una vez desplegado el complemento es responsabilidad del desarrollador adaptar el videojuego al complemento para que funcione correctamente en línea. Los pasos para seguir se detallan a continuación:

1. Ejecutar el archivo *Update assemblies in Unity project.bat* mostrado en la figura 22. Esto compilará las capas de conexión y mensajería en una librería y se copiarán a una carpeta que usa el proyecto de Unity de ejemplo.

[53] https://mobaxterm.mobatek.net/
[54] https://www.mono-project.com/download/stable/#download-lin

2. Copiar la carpeta *Unity\AshNet\Assets\Ashkatchap* que se muestra en la figura 23 en cualquier lugar dentro de la carpeta *Assets* de un proyecto Unity, como se ve en la figura 24.
3. Crear una nueva escena que se cargará cuando se pierda la conexión en línea o se detenga el servidor. En dicha escena añadir un `GameObject` que contenga los componente `NetController`, `NetServer` y `NetClient`, pudiendo también incluir el componente `NetControllerHUD` mientras no se disponga de una interfaz en el juego que interactúe con `NetController`. Configurar los componentes como se ve en la figura 25.
4. Crear un `prefab` a partir de un `GameObject` con un objeto controlable por el jugador y una cámara en su interior y configurarlo de modo que los componentes cuya función es mover el `GameObject` y la cámara estén desactivados. Configurar el componente `NetIdentity` para que, en caso de que ser el propietario del `GameObject`, se activen los anteriores elementos. La figura 26 muestra el estado final del `GameObject`.
5. Arrastrar el `prefab` del punto anterior al campo `PlayerActor` indicado en la figura 27.
6. Por último, agregar un `GameObject` en algún lugar de la escena y añadirle el componente `NetSpawnPoint` como se puede ver en la figura 28. El lugar donde se encuentra este `GameObject` será el lugar donde se instancie el anterior `prefab` cuando un jugador se conecte.

A partir de este punto ya es posible jugar en línea con otros jugadores.

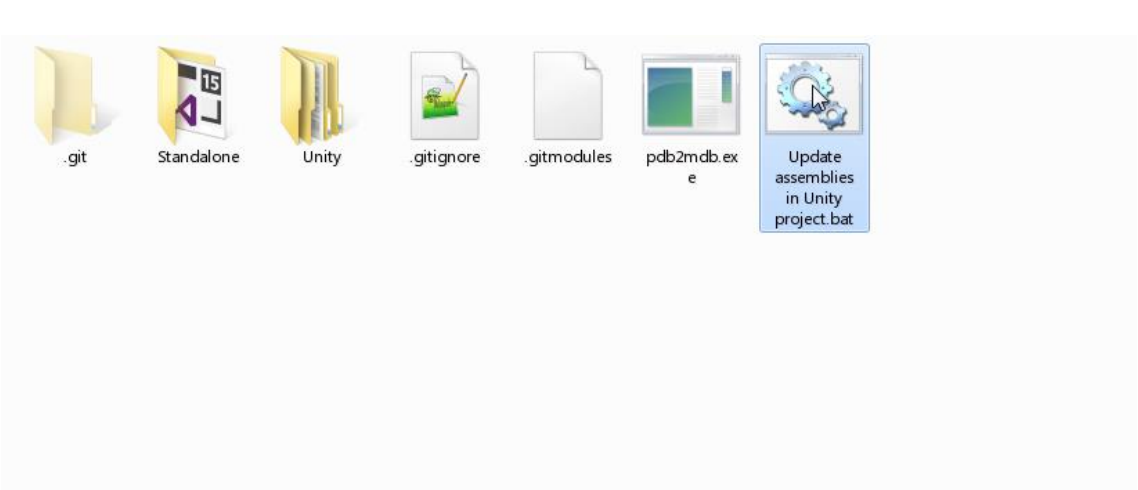

Figura 22 Archivo *Update assemblies in Unity project.bat*.

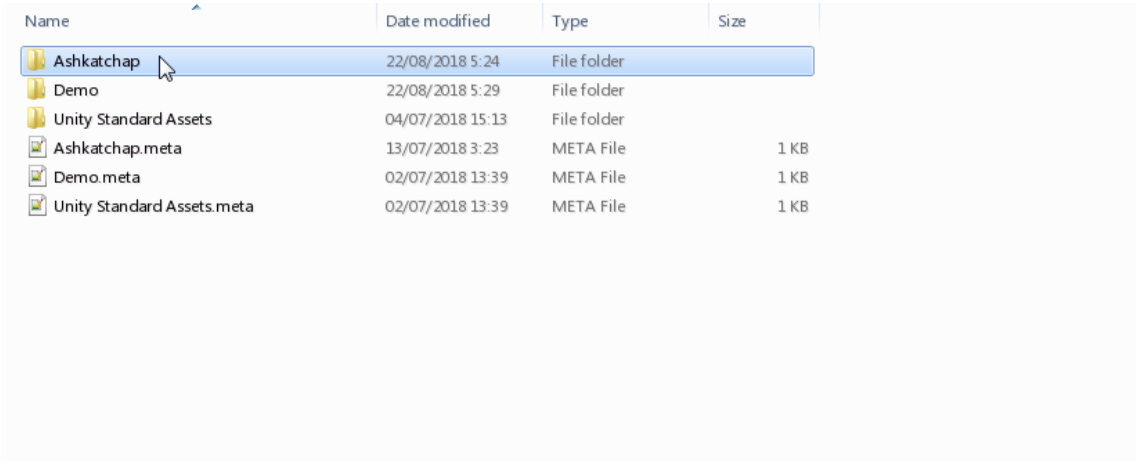

Figura 23 Carpeta con el contenido del complemento.

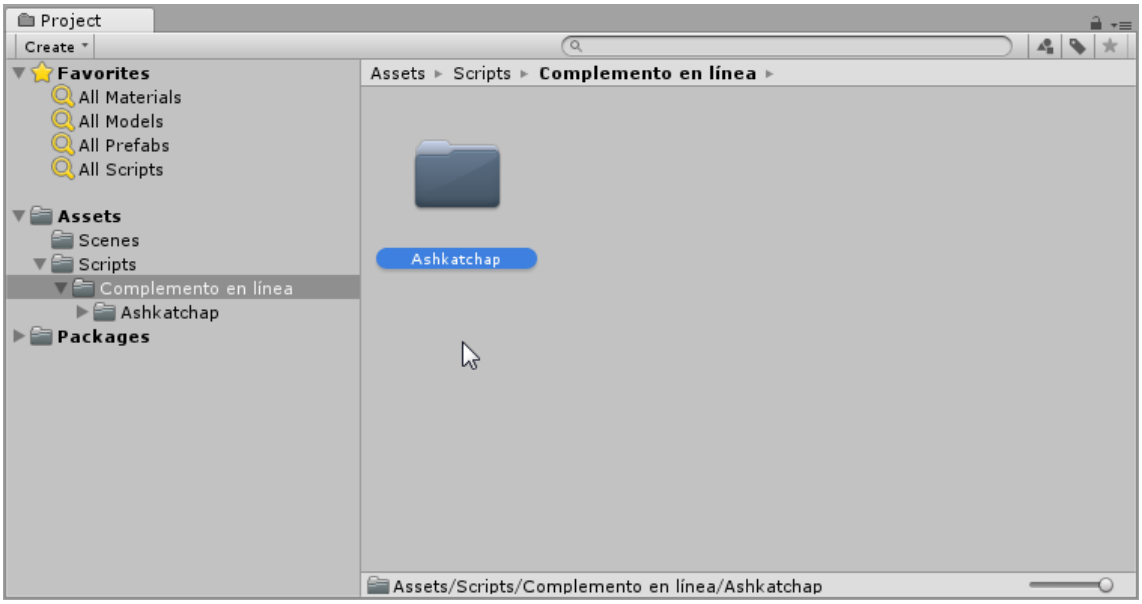

Figura 24 Ventana del proyecto en Unity mostrando el complemento.

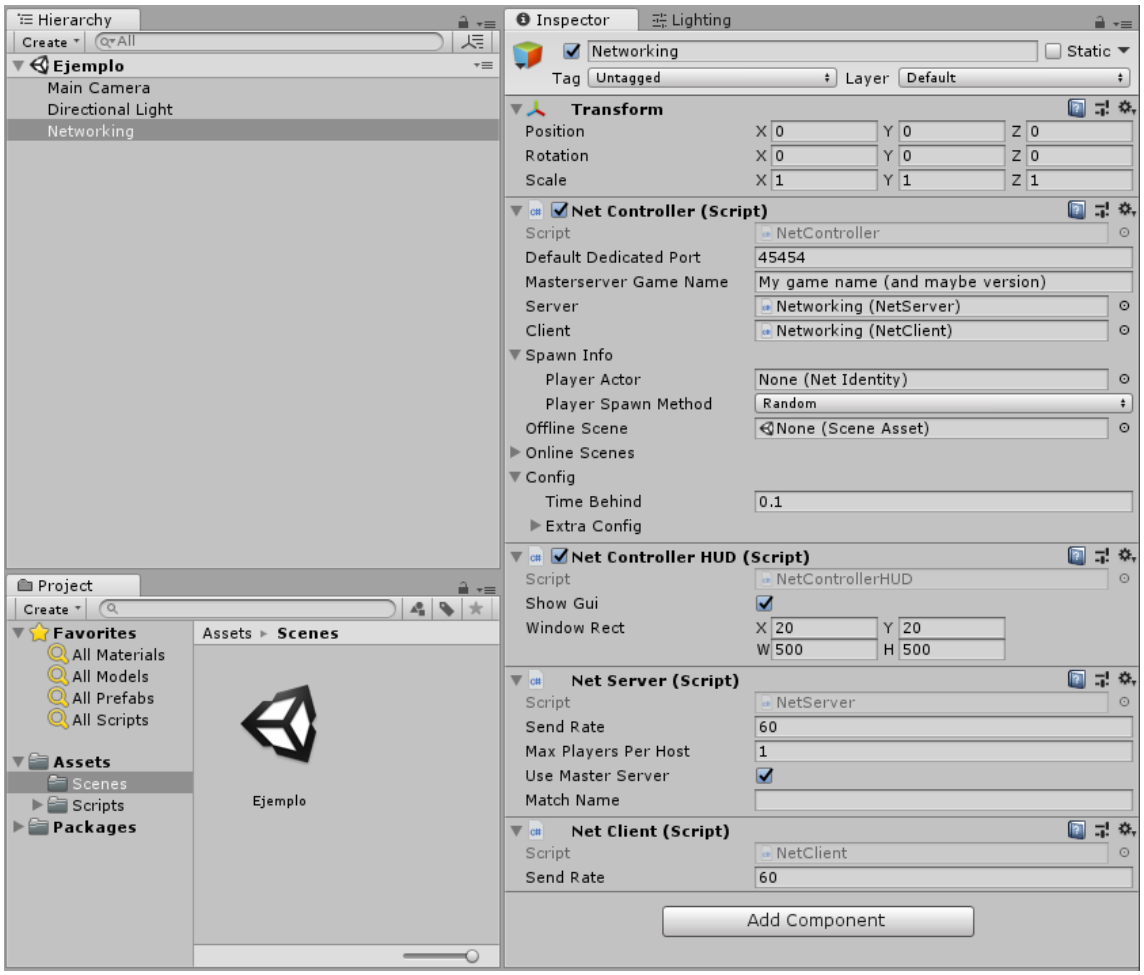

Figura 25 Configuración básica de los componentes.

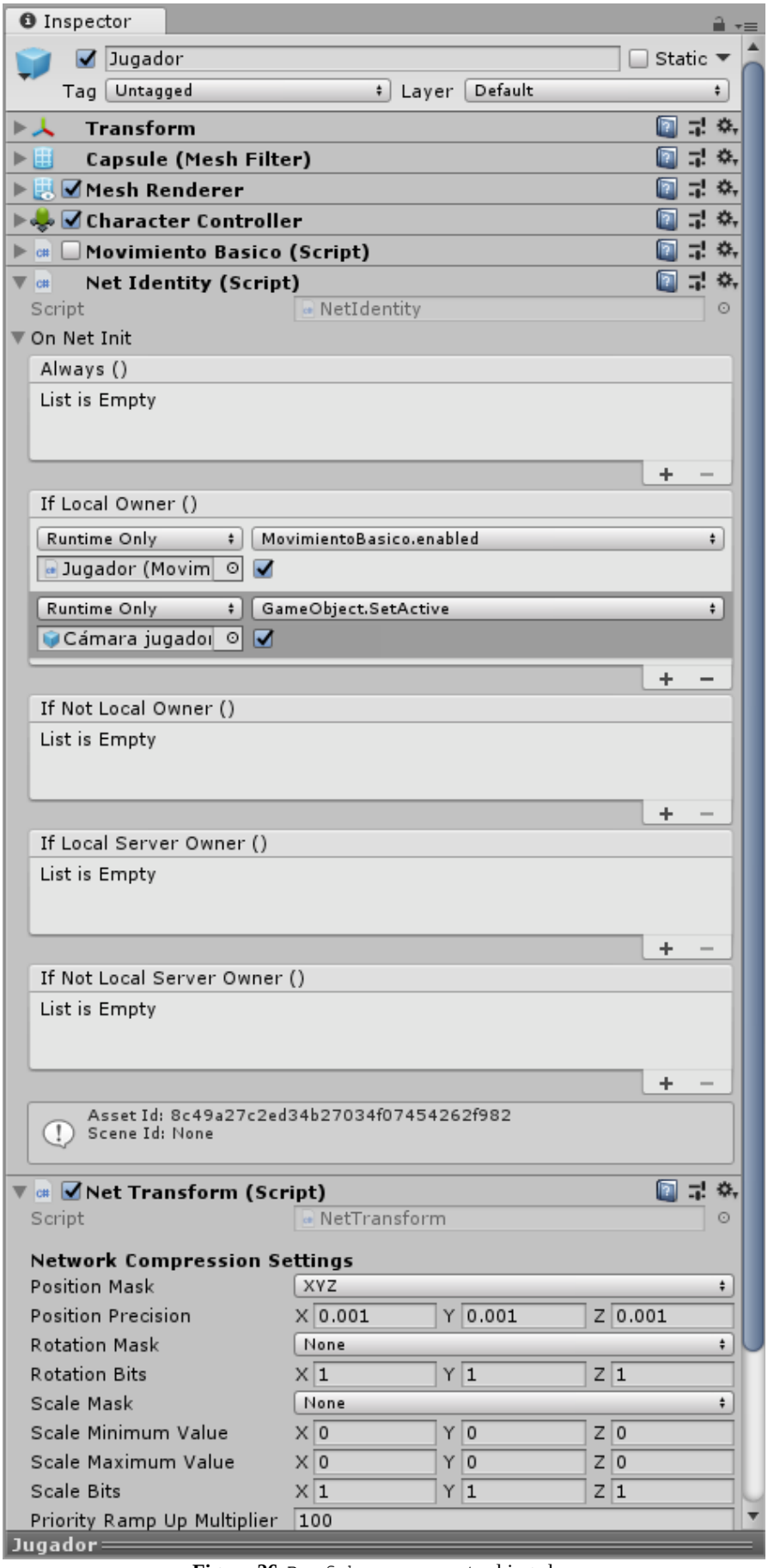

Figura 26 *Prefab* que representa el jugador.

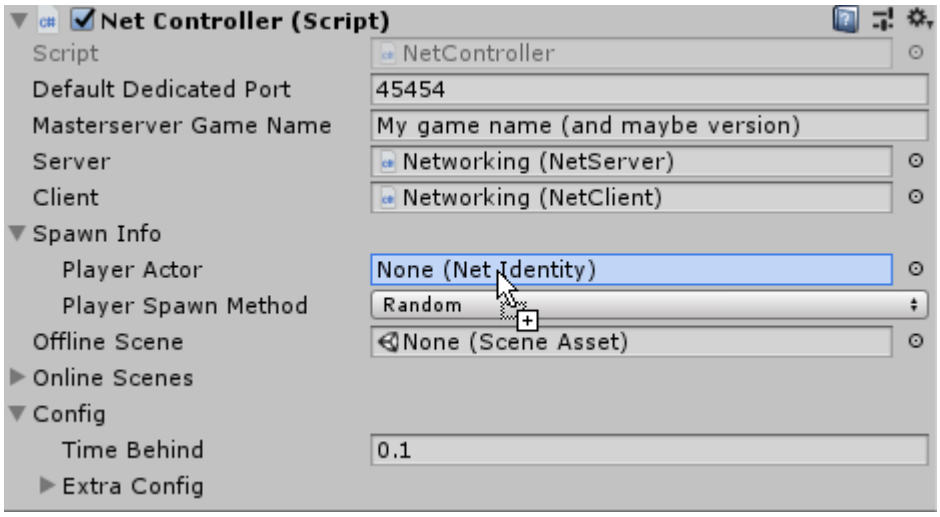

Figura 27 Configurando un *prefab* para actuar como actor de los jugadores.

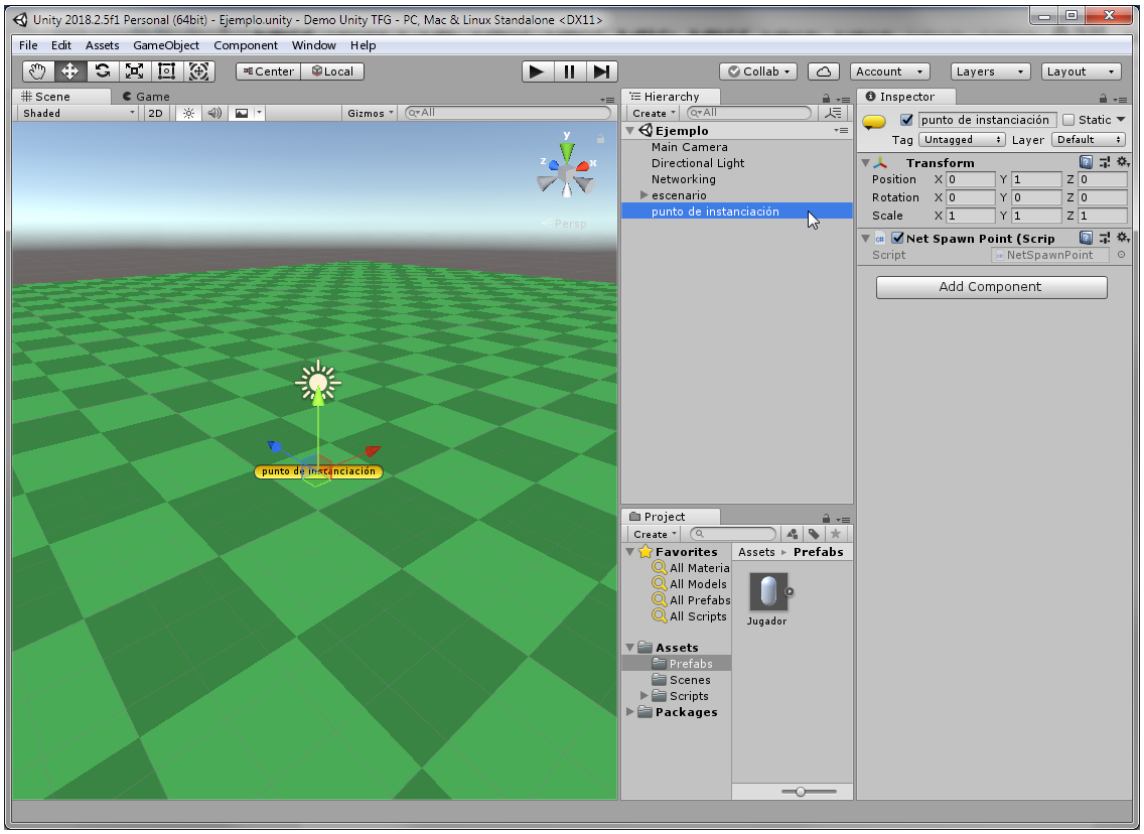

Figura 28 Añadiendo punto de instanciación para el *prefab* creado para cada jugador.

9. Pruebas

Para asegurar la calidad del producto se ha realizado mayoritariamente pruebas unitarias. Además, se han realizado varias pruebas que miden la velocidad de transferencia y consumo de recursos para poder medir las optimizaciones realizadas y comprobar si han sido efectivas. Todas las pruebas unitarias escritas para este proyecto se pueden observar en las figuras 29, 30, 31, 32 y 33.

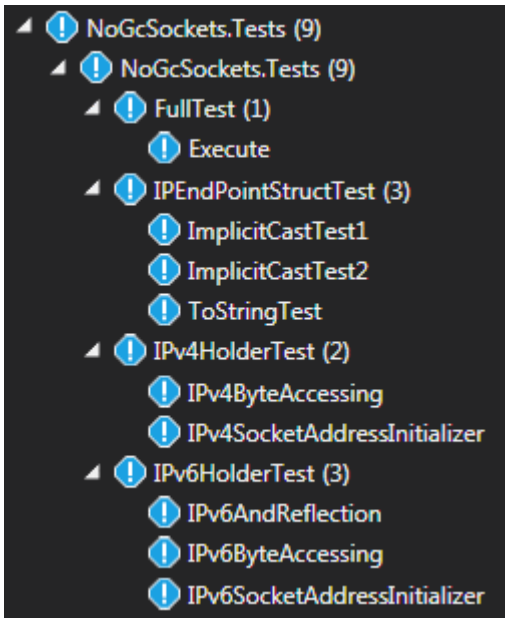

Figura 29 Pruebas unitarias para NoGcSockets.

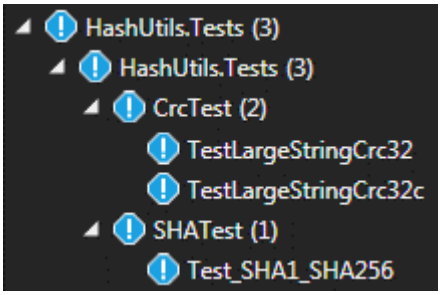

Figura 30 Pruebas unitarias para HashUtils.

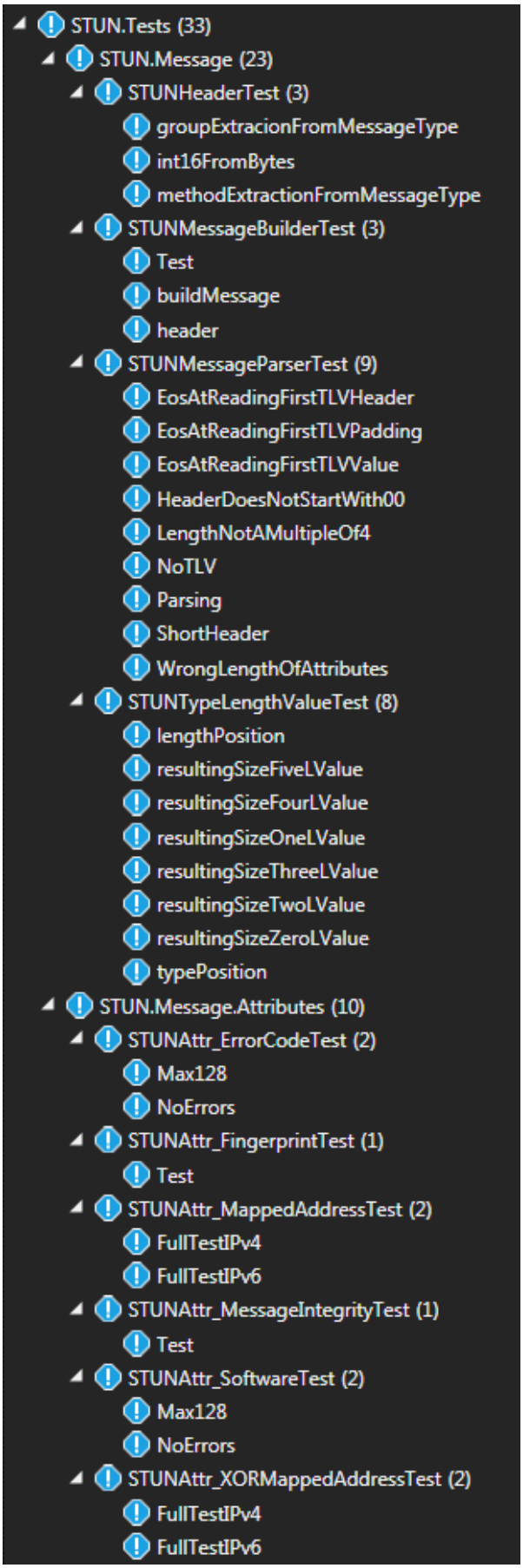

Figura 31 Pruebas unitarias para STUN.

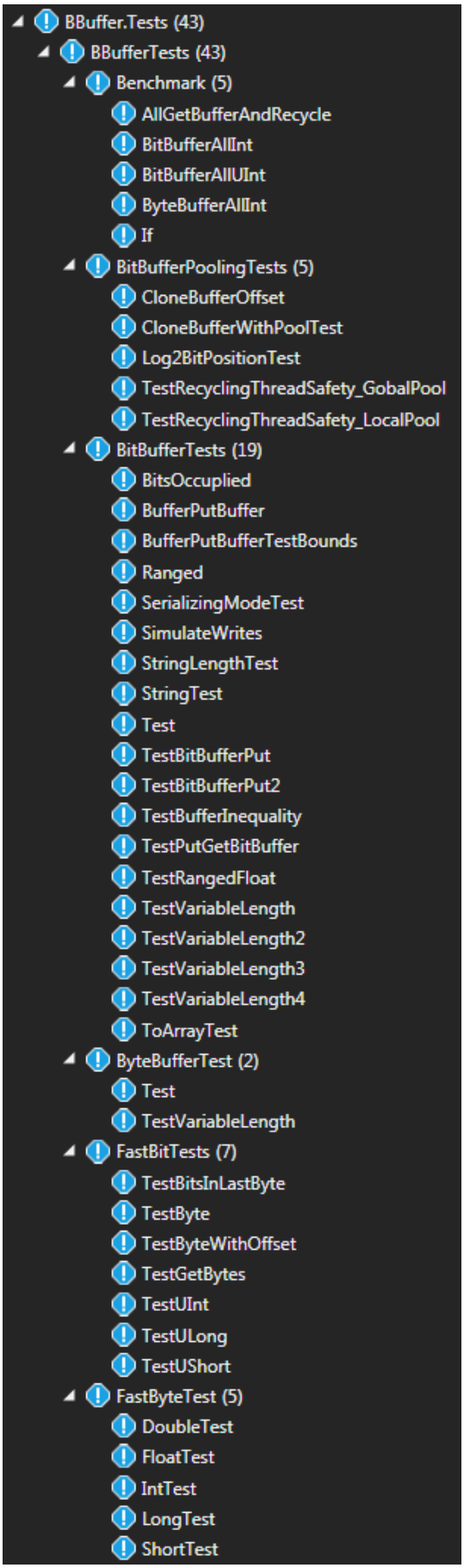

Figura 32 Pruebas unitarias para BBuffer.

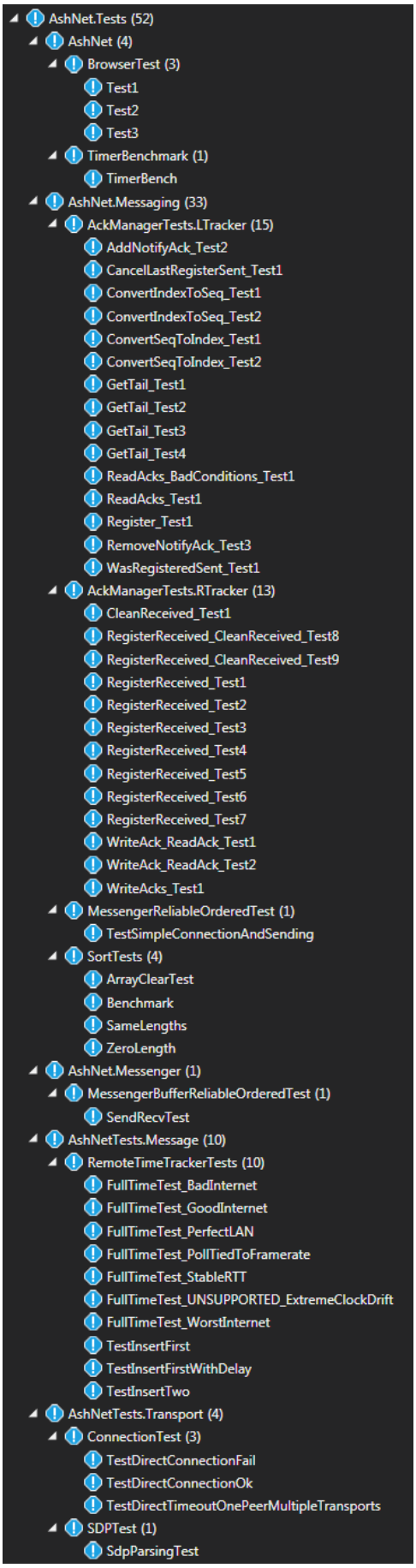

Figura 33 Pruebas unitarias para AshNet, capas de transporte y mensajería.

10. Conclusiones

Se ha obtenido la mayoría de los objetivos deseados:

- Se ha logrado optimizar el ancho de banda y CPU, pero no el uso de la memoria.
- Se ha logrado sincronizar el juego de un modo visualmente fluido.
- Se ha logrado dar soporte a todas las plataformas ofrecidas por Unity.
- Se ha facilitado la opción de extender el complemento gracias al diseño del complemento.
- Es posible introducir y alterar la lógica de la sincronización del estado videojuego con facilidad, aunque todavía se puede mejorar más.
- Se ha expuesto información interna que otros complementos ocultan, como acceso al sistema de acuse de recibo de los paquetes o conocimiento del reloj remoto en lugar de únicamente del reloj sincronizado entre todas las máquinas.

En cambio, los siguientes objetivos no se han alcanzado:

- La tolerancia a fallos en la red es muy limitada, esto se debe a la necesidad de más pruebas unitarias y a la simplificación del código durante el desarrollo con la intención de hacer robustas partes en el complemento cuando este funcionara como se esperaba.
- No se ha comercializado el complemento ya que requiere de más tiempo para mejorar el complemento, tal y como indicaron los participantes de la prueba sobre el segundo MVP.

Durante el desarrollo se ha cometido el error de introducir el uso de múltiples hilos para mejorar el uso de los recursos de la CPU. Esto supuso una menor productividad durante el desarrollo al hacer más complejo el código, por lo que se decidió dejar de usar múltiples hilos a excepción de uno, usado para recibir paquetes del socket. Esta decisión ayudó más adelante ya que la plataforma WebGL no soporta usar múltiples hilos, lo que hubiera complicado el diseño del código para permitir ser ejecutado con en un hilo o múltiples hilos.

Durante este proyecto se ha aprendido todo lo relacionado con WebRTC y comunicación P2P entre navegadores, las tecnologías que WebRTC usa requiriendo comprender los RFC que las describen. Ha sido necesario comprender detalles de Unity, concretamente de la implementación interna de componentes como `Animator`. Se ha investigado de múltiples protocolos de comunicación usados por otros videojuegos y protocolos no especializados en videojuegos, además de analizado y usado librerías y complementos similares o de la competencia con la intención de diseñar el protocolo usado en la capa de mensajería del complemento, aprendiendo de todos ellos.

Aunque no se ha podido poner en práctica en este trabajo, también se ha aprendido múltiples métodos de sincronización del estado de un videojuego. Este complemento usa uno de los métodos más sencillos que hay, pero se ha aprendido otros, cada uno con sus propias ventajas e inconvenientes.

Por último, la necesidad de interactuar con los participantes de los experimentos realizados durante el desarrollo y que estos interactuaran con el código ha supuesto una importante experiencia positiva.

10.1. Problemas y desafíos encontrados

La implementación de la API WebRTC en C# ha sido la parte más complicada llevada a cabo en este TFG debido a la cantidad de protocolos relacionados entre sí, la multitud de pequeños detalles en la implementación de estos y la existencia de errores de implementación en los navegadores web debido a que WebRTC todavía es una tecnología en desarrollo.

BouncyCastle no soporta DTLS 1.2, sólo 1.0. Esto ha hecho necesario investigar una solución, requiriendo investigar el protocolo DTLS junto con cómo ICE establece quién actúa como servidor y quién como cliente en la conexión DTLS, hasta finalmente encontrar cómo modificar el SDP usado para iniciar la conexión WebRTC para que una plataforma de escritorio, que usa BouncyCastle, siempre actúe como servidor y pueda usar DTLS 1.0.

El protocolo SCTP se basa en el envío de mensajes de distintos tipos, con distintas funcionalidades. Dos de estos tipos son *HeartBeat* y *HeartBeat ACK*, cuyo objetivo es indicar que el otro extremo de la conexión sigue funcionando y por tanto la conexión tiene que mantenerse abierta. Cada cierta cantidad de segundos se envía un mensaje *HeartBeat* y se espera la recepción de un mensaje *HeartBeat ACK*. De igual modo, cuando se recibe un mensaje *HeartBeat* se debe responder de inmediato con el mensaje *HeartBeat ACK*. La recepción de mensajes *HeartBeat* y *HeartBeat ACK* reinician un contador que, si alcanza cierto valor, cierra la conexión por *timeout*. El problema con este protocolo es que el envío de estos mensajes está muy espaciado en el tiempo y su pérdida supone el cierre de la conexión. SCTP también implementa *ACK* para mensajes del tipo de datos, pero estos mensajes no reinician el contador a pesar de indicar que la conexión sigue abierta. Además, algunas implementaciones sólo usan el mensaje *HeartBeat ACK* para reiniciar el contador. Todo esto resulta en la pérdida de una conexión por *timeout* en menos de un minuto si se pierden alrededor de un 10% de paquetes o más en la red. Este problema sólo existe en la plataforma WebGL ya que para este complemento se ha implementado SCTP teniendo este problema en cuenta, usando todos los mensajes *ACK* para reiniciar el contador.

Durante el desarrollo no se hizo uso extensivo de pruebas unitarias. A mitad del desarrollo empezaron a aparecer problemas en la librería BBuffer, corrompiendo el búfer al escribir o leer en el de forma incorrecta. A partir de este momento se empezó a usar de forma extensiva pruebas unitarias arreglando así muchos problemas, pero por falta de tiempo no fue posible hacer lo mismo en muchas partes que ya estaban implementadas.

10.2. Relación del trabajo desarrollado con los estudios cursados

El trabajo realizado hace uso de tecnologías bien conocidas en informática desde hace muchos años, concretamente comunicación en red mediante el uso de protocolos. Se ha extendido en este conocimiento al hacer uso de WebRTC y el enfoque del trabajo en las aplicaciones en tiempo real. De las asignaturas estudiadas durante el grado las siguientes tienen especial relación con este trabajo:

Concurrencia y sistemas distribuidos
Necesario para las partes del programa que usaban múltiples hilos y para la sincronización del reloj entre los participantes de una conexión ya que un apartado de la asignatura se centró en la sincronización de relojes en sistemas distribuidos.

Diseño de software
Se ha puesto en práctica estructuras de código y técnicas de programación aprendidas en la asignatura, tratando de realizar el código más mantenible posible mediante técnicas como *clean code*.

Redes de computadores
La mayoría del complemento hace uso de los conocimientos aprendidos en esta asignatura al tener relación directa.

Ingeniería del software
En esta asignatura se aprendió ha diseñar software mediante diagramas, técnica usada en este trabajo y que ha guiado todo el desarrollo.

Tecnología de sistemas de información en la red
Asignatura que imparte conocimientos sobre el intercambio de información por internet y los pone en práctica.

Introducción a la programación de videojuegos
Esta asignatura introduce a gran cantidad de aspectos de los videojuego, permitiendo tener una visión amplia sobre el desarrollo de estos, ayudado a evitar la realización de errores de diseño por desconocimiento. Además, ha guiado a la hora de decidir las necesidades que el complemento necesitaba cubrir.

Mantenimiento y evolución de software
Los conocimientos de esta asignatura han permitido la realización de pruebas unitarias y la interacción con ellas.

Proyecto de ingeniería de software
El marco de emprendimiento tomado por este TFG es gracias a esta asignatura. Esta asignatura ha indicado los pasos a seguir en el desarrollo usando un enfoque ágil.

10.3. Trabajos futuros

La API de WebRTC cuenta con varias diversas limitaciones:

- No se permite la selección del puerto que se usará para una conexión.
- Cada conexión hace uso de un puerto distinto.
- Es obligatorio usar encriptación

Mientras que deshabilitar la tercera limitación crearía una incompatibilidad, las dos primeras limitaciones no están presentes en este complemento. Sería interesante hacer uso de permitir usar siempre el mismo puerto en un servidor, siendo interesante investigar de qué formas el levantamiento de ciertas restricciones de WebRTC pueden beneficiar al complemento.

Se deseaba implementar todas las características listadas en el punto de desarrollo de la solución propuesta, pero esto ha sido imposible por falta de tiempo. Se espera seguir añadiendo características, tanto las listadas anteriormente como nuevas (integración de nuevas API de Unity, como el sistema de trabajos), en el futuro.

Se pretende trabajar en la baja tolerancia a fallos en la red y baja resistencia a ataques externos, siendo esto una prioridad una vez el complemento tenga más funcionalidades y sea más estable.

Este complemento pretendía ser comercializado durante el desarrollo del TFG, pero por la complejidad del mismo y falta de tiempo, esto no ha sido posible. Se espera comercializar el complemento en el futuro cercano, pero será necesario dedicar más tiempo del esperado en su desarrollo.

Durante el desarrollo se implementó el complemento como una librería y una capa de Unity independientes entre sí. Cuando comenzó el desarrollo de la capa de Unity se unificó todo como un complemento único, pero más adelante se volvió a separar en una librería y la capa de Unity. Esta librería, que es independiente de Unity, podría ser usada con otros motores de videojuegos que soporten C#, como Godot[55], Monogame[56] o Xenko[57], además de en otras aplicaciones no relacionadas con videojuegos. Esta es una opción que se quiere explorar más adelante.

El protocolo usado en la capa de mensajería puede ser optimizado para reducir la cantidad de ancho de banda invertido en cabeceras. Se continuará trabajando en reducir la cantidad de bytes usados por las cabeceras en la medida de lo posible.

Se mejorará la tolerancia a fallos de red, concretamente la tolerancia a paquetes con contenido arbitrario que pudiera provocar un funcionamiento no deseado. Este problema ocurre cuando se corrompe un paquete en la red, que es muy extraño, o cuando se recibe un ataque. Es necesario añadir medidas contra este problema para evitar graves problemas, como el que ocurrió a los desarrolladores del videojuego RUST, que muestran en su blog como el envío de paquetes en masa, con contenido especialmente preparado para lanzar excepciones en el servidor, daba problemas y, como no tenían acceso al código fuente de la librería de *networking*, no podían repararlo (26).

Por último, este complemento será usado en un videojuego en línea que se desarrollará tras la finalización del TFG. Este proceso contribuirá en el desarrollo del complemento al ser expuesto a una situación real.

[55] https://godotengine.org/
[56] http://www.monogame.net/
[57] https://xenko.com/

11. Agradecimientos

En primer lugar, quiero agradecer a mi familia, en especial a mis padres, por el apoyo constante todo este tiempo.

Gracias a Ramón Mollá y a Patricio Letelier por guiarme durante la elaboración de este TFG y tener paciencia con el lento desarrollo de la memoria.

Gracias Miguel Ángel García por trabajar conmigo tanto tiempo y darme la oportunidad de hacer videojuegos y no demos técnicas.

Finalmente, gracias a los participantes de los experimentos por dedicar tiempo en probar el complemento y su interés en ella, especialmente a Joseph Friend, Brian Hernandez y Aviel Basin.

Gracias.

12. Referencias

1. **House, Brandi.** Evolving multiplayer games beyond UNet. *Unity Blog.* [En línea] 2 de agosto de 2018. https://blogs.unity3d.com/2018/08/02/evolving-multiplayer-games-beyond-unet/.

2. **nxrighthere.** Benchmark Results. *GitHub.* [En línea] 22 de octubre de 2018. https://github.com/nxrighthere/BenchmarkNet/wiki/Benchmark-Results.

3. **Aldridge, David y LLC, Bungie.** I Shot You First: Networking the Gameplay of HALO: REACH. *GDC Vault.* [En línea] 2011. https://www.gdcvault.com/play/1014345/I-Shot-You-First-Networking.

4. **Bernier, Yahn W.** Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. *VALVe Developer Community.* [En línea] 2001. https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization.

5. **Fiedler, Glenn.** Snapshot Interpolation. Interpolating between snapshots of visual state. *Gaffer On Games.* [En línea] 30 de noviembre de 2014. https://gafferongames.com/post/snapshot_interpolation/.

6. **STEAMWORKS.** Steam Networking. *STEAMWORKS.* [En línea] https://partner.steamgames.com/doc/features/multiplayer/networking.

7. **Kurose, James F. y Ross, Keith W.** *Redes de computadoras, Un enfoque descendente.* 5. s.l. : PEARSON, 2010. págs. 229-236.

8. **Unity.** Multiplayer Overview. *Unity Documentation.* [En línea] 31 de julio de 2018. https://docs.unity3d.com/Manual/UNetOverview.html.

9. **Photon.** Binary Protocol . *Photon.* [En línea] https://doc.photonengine.com/en-us/pun/current/reference/binary-protocol.

10. **Farris, Brent.** Forge Networking Jumpstart. *YouTube.* [En línea] 21 de marzo de 2017. https://www.youtube.com/playlist?list=PLm1w78-UUlMIi5Vfwy6ckJQIQMHMT-QS5.

11. **Auriemma, Luigi y Ferrante, Donato.** Game Engines: A 0-day's Tale. *ReVuln.* [En línea] 17 de mayo de 2013. http://revuln.com/files/ReVuln_Game_Engines_0days_tale.pdf.

12. —. Exploiting Game Engines For Fun & Profit. *ReVuln.* [En línea] mayo de 2013. http://revuln.com/files/Ferrante_Auriemma_Exploiting_Game_Engines.pdf.

13. —. Multiplayer Online Games Insecurity. *ReVuln.* [En línea] 12 de marzo de 2013. https://media.blackhat.com/eu-13/briefings/Ferrante/bh-eu-13-multiplayer-online-games-ferrante-slides.pdf.

14. **Dunn, Fletcher.** Denial of Service Mitigation. *YouTube.* [En línea] 26 de julio de 2018. https://www.youtube.com/watch?v=2CQ1sxPppV4.

15. **Rosenberg, J., y otros.** Session Traversal Utilities for NAT (STUN). *Internet Requests for Comments.* [En línea] octubre de 2008. https://tools.ietf.org/html/rfc5389.

16. **Rescorla, E., y otros.** Datagram Transport Layer Security. *Internet Requests for Comments.* [En línea] abril de 2006. https://tools.ietf.org/html/rfc4347.

17. **Rescorla, E., y otros.** Datagram Transport Layer Security Version 1.2. *Internet Requests for Comments.* [En línea] enero de 2012. https://tools.ietf.org/html/rfc6347.

18. **R. Stewart, Ed.** Stream Control Transmission Protocol. *Internet Requests for Comments.* [En línea] septiembre de 2007. https://tools.ietf.org/html/rfc4960.

19. **Stewart, R., y otros.** Stream Control Transmission Protocol (SCTP). Partial Reliability Extension. *Internet Requests for Comments.* [En línea] mayo de 2004. https://tools.ietf.org/html/rfc3758.

20. **Jesup, R., y otros.** WebRTC Data Channels. draft-ietf-rtcweb-data-channel-13.txt. *Internet Requests for Comments.* [En línea] 4 de enero de 2015. https://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-13.

21. —. WebRTC Data Channel Establishment Protocol. draft-ietf-rtcweb-data-protocol-09.txt. *Internet Requests for Comments.* [En línea] 4 de enero de 2015. https://tools.ietf.org/html/draft-ietf-rtcweb-data-protocol-09.

22. **Fette, I., y otros.** The WebSocket Protocol. *Internet Requests for Comments.* [En línea] diciembre de 2011. https://tools.ietf.org/html/rfc6455.

23. **Booth Simpson, Zachary.** A Stream-based Time Synchronization Technique For Networked Computer Games. *Mine-Control.* [En línea] 1 de marzo de 2000. http://www.mine-control.com/zack/timesync/timesync.html.

24. **Photon.** Performance Tips . *Photon.* [En línea] https://doc.photonengine.com/en-us/onpremise/current/performance/performance-tips.

25. **Parker, Alex.** How to Optimize Lidgren. *GitHub.* [En línea] 4 de agosto de 2015. https://github.com/lidgren/lidgren-network-gen3/wiki/Optimization.

26. **Facepunch Studios.** ULink DDos Attacks . *Blog/News - Rust.* [En línea] 27 de diciembre de 2013. https://rust.facepunch.com/blog/ulink-ddos-attacks/.

13. Anexos

13.1. Glosario de términos

Animación
Colección de valores espaciados en el tiempo que cuando pueden ser interpretados por un `Animator`, animando así valores en objetos de Unity a lo largo del tiempo.

Animator
Componente de Unity que implementa una máquina de estados. Cada estado representa una animación, varias animaciones u otra máquina de estados. Mediante el uso de variables editables desde scripts se puede controlar la máquina de estados consiguiendo así animar un objeto.

Asset Store
Tienda virtual oficial de complementos para Unity. Contiene scripts, modelos 3D, imágenes, audio, *shaders* y más. Permite que desarrolladores puedan vender sus productos, pasando antes por un escrutinio del mismo que dura alrededor de un mes. Unity se queda con un 25 % del dinero ganado.

Callback
Puntero a una función que se guarda en una variable y permite invocar la función usando esta variable. *Callback* es concretamente una función que se pasa a otra como un parámetro para que sea invocada una vez termine la ejecución de la función que usualmente es asíncrona. En C# se implementa mediante la API `delegate`.

CCU
Cantidad de usuarios concurrentes. Indica la cantidad de usuarios que están concurrentemente usando un recurso. Usado en el ámbito de servidores para videojuegos, indica la cantidad de gente que está jugando en línea al mismo tiempo usando dichos servidores.

CIL (Infraestructura de lenguaje común)
También conocido como MSIL, es el código al que se compila C# usando .NET Framework, entre otros. CIL es un lenguaje de ensamblador basado en una máquina *stack*. Código compilado en CIL puede tomar forma de librerías o de ejecutables.

Compresión diferencial
Compresión diferencial es una función de compresión que, dado un primer valor que se quiere comprimir y un segundo valor, retorna un nuevo valor que es la diferencia entre el primer y el segundo valor. Este nuevo valor no tiene por qué ser menor que el primer valor, pero si la diferencia entre el primer y el segundo valor es pequeña, el nuevo valor será pequeño. Esta función de compresión es útil cuando el primer valor no cambia bruscamente y se usa como segundo valor una versión más antigua del primer valor.

`GameObject`
Entidad de Unity que actúa como contenedor de *`Transform`* y otros componentes. Está contenido dentro de una escena o en un *`prefab`*, pero a diferencia de *`Transform`*, no tiene noción de jerarquía con respecto a otros *`GameObject`*.

Hacking
En este TFG el término se usa para representar la acción de intentar explotar un sistema de forma fraudulenta o con intención de romperlo por cualquier motivo.

Host migration
Técnica usada en arquitecturas de red en las que múltiples jugadores están conectados a otro jugador que actúa como servidor. Cuando el jugador que actúa como servidor desaparece, la técnica *Host migration* convierte a uno de los jugadores restantes en el nuevo servidor y el resto de los jugadores se conecta a él.

Lag compensation
Técnica usada para ocultar la presencia de latencia en la red durante un juego. Consiste en que el servidor tome decisiones con respecto al *input* indicado por el jugador de forma remota usando el estado que el jugador veía en el momento que realizó el *input*. Por ejemplo, si el *input* es hacer clic cuando el centro de la pantalla de jugador muestra el avatar de otro jugador, debido a la latencia de la red, para cuando el *input* llegue al servidor, el estado del juego habrá cambiado y el jugador, desde el punto de vista del servidor, no habrá hecho clic sobre el avatar del otro jugador. Para corregir esto, el servidor retrocede el estado del videojuego al momento en el que el *input* se realizó en el jugador y entonces evalúa el clic, que ahora sí, estará realizado sobre el avatar de otro jugador.

LAN discovery
Proceso por el cual se descubre la dirección de servidores que se están ejecutando en la misma LAN que nos encontramos.

LOD
Nivel de detalle (*Level Of Detail*). Se usa para expresar que determinados ciertos parámetros, como por ejemplo distancia entre dos objetos o tiempo transcurrido desde un evento, determinan la importancia de un objeto, afectando a su calidad, detalle o precisión. En este TFG se usa para expresar que el estado de objetos que se consideran poco importantes es sincronizado en menor medida, o incluso se deja de sincronizar.

Matchmaking
Técnica usada por un servidor maestro al que un jugador se suscribe para jugar en línea. El servidor agrupa jugadores con un nivel de habilidad similar en un mismo servidor con la intención de que la experiencia de juego sea ideal.

MVP
Producto mínimo viable. Representa un producto que implementa lo mínimo para poder considerarse viable para un objetivo determinado, como ser usado o comercializado.

NAT hole punching
Técnica usada para solventar el problema que previene iniciar un servidor tras una NAT mediante el intercambio de paquetes entre dos direcciones, creando una conexión. Es necesario el uso de un tercero para llevar a cabo la técnica.

NPAPI
Netscape Plugin Application Programming Interface, es una API que permite crear complementos para navegadores web con el objetivo de mostrar tipos concretos de contenido mime que no soporta el navegador web. Estos complementos pueden interactuar directamente con la máquina (deben de encontrarse instalados en la misma). Debido a problemas de seguridad los navegadores web más importantes actualmente han mostrado su intención de dejar de soportarlos.

`Prefab`
`GameObject` guardado como un *asset* de Unity, permitiendo que no esté atado a una escena.

RPC
Remote Procedure Call, técnica para ejecutar funciones en máquinas remotas.

Serializable
Habilidad de transformar un objeto en texto y de deshacer la transformación para recuperar el objeto.

Servidor *relay*
Servidor que actúa como intermediario en una comunicación entre participantes que no pueden comunicarse directamente entre ellos. Aunque es posible usar un servidor *relay* pudiendo establecer una conexión directa, no siempre es recomendable. Los servidores de este tipo tienen un gran uso de ancho de banda, requiriendo de una inversión económica para ponerlos en marcha. La situación estratégica de estos servidores y una comunicación directa y rápida entre ellos puede resultar en que el uso de un servidor *relay* supone tener menor latencia que en una conexión directa.

`Transform`
Componente de Unity que siempre se encuentra en un `GameObject`. Provee al `GameObject` de una estructura jerárquica con otros `GameObject` y de una matriz que representa su posición, rotación y escala en el espacio.

13.2. Muestras de código

Archivo *asset-store-parser.php* usado para contabilizar comentarios en la Asset Store en el punto Estudio de mercado.

```
<form method="GET">
<textarea name="webs" size="100" rows="20" cols="100"
placeholder="https://assetstore.unity.com/packages/..."><?php if
(isset($_GET["webs"])) echo htmlentities($_GET["webs"])?></textarea>
<br/>
<input type="submit"/>
</form>
```

```php
<?php
if (!isset($_GET["webs"])) return;
set_time_limit(0);

foreach (explode("\r\n", $_GET["webs"]) as $web) {
	echo "Procesando asset " . $web . '<br/>';
	for ($page = 1; ProcessWeb($web . '/reviews/?page=' . $page .
'&sort_by=recent', $datos); $page++);
}

echo "<table>";
foreach ($datos as $user => $date) {
	echo "<tr><td>" . $user . "</td><td>" . $date . "</td></tr>";
}
echo "</table>";


function ProcessWeb($url, &$datos) {
	$html = CargaWebCurl($url);

	if (false === strpos($html, '<div
id="Product/ReviewController">')) return false;
	if (false != strpos($html, 'There are no reviews yet, be the
first to post one!')) return false;

	$comentarios = entre1y2($html, '<div
id="Product/ReviewController">', '<!-- Login/Logout Passively -->');

	if (preg_match_all('@<div class="_2gdPy" data-reactid="[0-
9]+">([a-z0-9 ]+?)</div></div><div class="_3BI3F" data-reactid="[0-
9]+"><a class="bCYnm" href="/users/[0-9]+" target="_self"
rel="noopener noreferrer" data-reactid="[0-9]+">(.+?)</a>@im',
$comentarios, $matches)) {
		for ($i = 0; $i < count($matches[0]); $i++) {
			$datos[$matches[2][$i]] = $matches[1][$i];
		}
	}

	return true;
}

function entre1y2(&$que, $start, $fin){
	$p = strpos($que, $start) + strlen($start);
	$f = strpos($que, $fin, $p);
	return substr($que, $p, $f - $p);
}

function CargaWebCurl($url){
	echo 'Descargando: ' . $url . '<br/>';

	$cabeceras = array();
	$cabeceras[] = 'User-Agent: Comment crawler';
	$cabeceras[] = 'Accept: text/html;q=0.9,*/*;q=0.8';
	$cabeceras[] = 'Accept-Language: en-US,en;q=0.5';
	$cabeceras[] = 'Connection: close';

	$ch = curl_init();
	curl_setopt($ch, CURLOPT_URL, $url);
	curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
	curl_setopt($ch, CURLOPT_HEADER, 0);
```

```
        curl_setopt($ch, CURLOPT_FOLLOWLOCATION, 1);
        curl_setopt($ch, CURLOPT_COOKIEFILE, "");
        curl_setopt($ch, CURLOPT_HTTPHEADER, $cabeceras);
        curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, 0);
        curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, 0);
        curl_setopt($ch, CURLOPT_ENCODING, '');

        return curl_exec($ch);
}
```