TecnoCampus

Centres universitaris adscrits a la

upf. Universitat Pompeu Fabra Barcelona

Grau en Disseny i Producció de Videojocs

Diseño y creación de una suite de protocolos de capa de red orientado a videojuegos

Jan Lopera Cano
Tutor: Rafael González Fernández

TecnoCampus Mataró-Maresme

Dedicatoria

Dedicado a todos los que enseñan con pasión aquello que aprendieron y que inspiran a tantas personas.

Agradecimientos

A mi familia y amigos por el apoyo que me han dado durante la realización de este proyecto, sin ellos habría sido imposible.

A mi tutor, Rafa, quien ha confiado en mí y ha apostado por un proyecto que parecía una locura.

A mi padre, por dejar aquellos libros de Redes Novell en casa de los que aprendí tanto cuando era pequeño.

Abstract

This project focuses on the design and development of a network protocol suite that can compete with established protocols, thus enabling the creation of a new network stack that is optimized for current network usage. Throughout the report, the design of the protocols is presented, their use is detailed, and the necessary software is developed for use in the well-known Unity game engine.

Resum

Aquest projecte se centra en el disseny i desenvolupament d'una suite de protocols de xarxa que sigui capaç de competir amb els protocols establerts; permetent, així, la creació d'una pila de xarxa nova que estigui optimitzada per a l'ús actual d'Internet. Al llarg de la memòria s'exposa el disseny dels protocols, es detalla el seu ús, i es desenvolupa el programari necessari per a ser utilitzat al conegut motor gràfic Unity.

Resumen

Este proyecto se centra en el diseño y desarrollo de una suite de protocolos de red que sea capaz de competir con los protocolos establecidos; permitiendo, así, la creación de una pila de red nueva que esté optimizada para el uso actual de la red. A lo largo de la memoria se expone el diseño de los protocolos, se detalla su uso, y se desarrolla el software necesario para ser usado en el conocido motor gráfico Unity.

Índice

Índice de figuras

Capítulo I: Introducción y objeto del proyecto

Aram Ter-Gazarian describe los grandes ordenadores de la década de los 40s en la Unión Soviética como máquinas tan capaces y avanzadas como sus homólogas del bloque occidental (Ter-Gazarian, 2014). Sin embargo, Yuri Revich y Valery Shilov, investigadores de la historia de la computación soviética, son más pesimistas en cuanto a esas declaraciones (Revich & Shilov, 2017). En su investigación relatan que gran parte de los artículos escritos acerca del tema utilizan como fuente a los propios desarolladores, que acostumbran a recordar el pasado con cierta nostalgia y con una visión demasiado subjetiva del estado real del mundo de la informática soviética en aquellos tiempos.

Si bien es cierto que ambos artículos difieren en cuanto al estado real del desarrollo a nivel informático de la URSS, en ambos casos sale señalada una decisión política que afectó de forma negativa a la innovación del bloque más al este del telón de acero: la creación del ES EVM, o el primer ordenador soviético clónico del IBM S/360. El desarrollo de este dispositivo supuso el fin de los esfuerzos del aparato ruso en la investigación y el desarrollo de dispositivos propios durante toda una década.

No fue hasta los 80s, cuando el estancamiento del mundo de la computación rusa era más que evidente, que se retomaron todas aquellas ideas planteadas en los años 60s y la brecha entre el bloque occidental y el oriental empezó a estrecharse.

En un mundo donde el uso de TCP/IP es incuestionable y todo un estándar en la industria y cuyo máximo desarrollo en esta área ha sido la creación de la poco adoptada (Fruhlinger, 2022) versión seis del protocolo ¿es posible que se esté viviendo una situación similar a la vivida en los años 70s en la Unión Soviética? ¿Hay quizá otras alternativas que se adapten mejor a la situación actual del mundo de la informática?

El objeto principal de este trabajo de fin de grado es plantear una alternativa al protocolo IP, prácticamente omnipresente en el Internet actual; desde un punto de

vista actual y moderno; y mediante el diseño de una suite de protocolos de capa tres y superior que permitan una mejor experiencia en el mundo del videojuego moderno.

Capítulo II: Objetivos

En este capítulo se pretende detallar los objetivos primarios y secundarios del proyecto, además de definir el alcance del proyecto y su público objetivo.

2.1 Objetivos del producto

Los objetivos del producto se dividen en objetivos primarios y secundarios. Los objetivos primarios conforman el núcleo del producto y definen las características que deben estar presentes en la versión final; en cambio, en los objetivos secundarios se establecen todos los objetivos que, sin dejar de ser importantes, tienen un menor peso en el resultado final del producto y pueden ser considerados elementos no esenciales.

2.1.1 Objetivos primarios

Se han determinado los siguientes objetivos primarios del proyecto.

1. Comunicar a dos clientes dentro de una misma red a través de la nueva suite de protocolos.
2. Comunicar a dos clientes de diferentes redes a través de la nueva suite de protocolos.
3. Permitir que Unity sea capaz de enviar y recibir información a través de la nueva suite de protocolos.
4. Ofrecer documentación detallada acerca de los nuevos protocolos.

Estos objetivos han sido convertidos en las siguientes tareas.

- Diseño de una suite de protocolos de red.
- Desarrollo de un controlador de protocolo.
- Creación de un SDK para Unity.
- Desarrollo de un juego simple multijugador.
- Desarrollo de un software de routing entre redes.

2.1.2 Objetivos secundarios

Se han determinado los siguientes objetivos secundarios:

1. Ofrecer encapsulación IP para poder utilizar redes convencionales.
2. Ofrecer un protocolo de anuncio de servicios de la red.
3. Ofrecer un protocolo de anuncio de partidas LAN.
4. Ofrecer un protocolo de anuncio de rutas dinámicas.

2.2 Público objetivo

El público objetivo del producto se puede clasificar, en un principio, en dos grandes perfiles diferenciados.

El primero es el de un administrador de sistemas que busque nuevas tecnologías en el ámbito de las redes Este perfil es un experto del sector y lo que busca es experimentar con tecnologías innovadoras, descubrir nuevos conceptos, y estar preparado para las posibles redes del futuro. En este perfil, también se puede incluir entusiastas del mundo de la tecnología que busquen estar en la vanguardia del desarrollo técnico. En ambos casos, el perfil es el de una persona con conocimiento en el área de la informática que no tiene miedo alguno en instalar drivers no firmados, o probar tecnologías que pueden causar inestabilidad en su dispositivo de preferencia.

El segundo es el de un desarrollador de juegos que busque una plataforma de red que le permita programar juegos en línea sin tener que lidiar con los problemas de la NAT, e incluso los problemas ocasionados al crear una sala de espera en LAN. Este perfil es el de un desarrollador que esté familiarizado con el uso de librerías de red de bajo nivel, además del motor gráfico Unity (o cualquier motor gráfico que pueda soportar una librería de .Net o una librería compilada mediante el uso de NativeAOT).

En ambos perfiles se requiere de un alto nivel de conocimiento técnico para poder utilizar y operar con este producto.

2.3 Alcance del proyecto

Este proyecto pretende entregar el diseño final de la especificación de los protocolos, además de un prototipo de estos, que en ningún momento está pensado para ser utilizado en un entorno de producción. El producto entregado es más una prueba de concepto de que se puede competir con los protocolos actuales, ofreciendo una serie de características novedosas, que un producto completamente acabado.

Capítulo III: Contexto y antecedentes

Joseph Carl Robnett Licklider, más conocido como Lick o J.C.R., publicó unos memorándums acerca de lo que él denominaba como “Galactic Network”. En estas notas se describía las nuevas interacciones sociales que una red global de redes interconectadas podría ofrecer, además de su visión personal en cuanto a cómo sería esta red de redes (Leiner, et al., 2009). Su punto de vista es muy similar a lo que hoy se conoce como Internet.

El primer intento de realizar este tipo de conexiones fue mediante el uso de una red de circuitos conmutados (sistema usado en las antiguas redes telefónicas convencionales), y no fue hasta 1964, año en que Leonard Kleinrock publicó el primer libro acerca de redes de paquetes conmutados, que los investigadores de DARPA decidieron cambiar a este tipo de topología de red (Leiner, et al., 2009).

3.1 Packet-Switched Networks

La invención de las redes de paquetes conmutados fue una revolución sobre las redes de circuitos conmutados, que son usadas en el sistema telefónico analógico. Esto es debido al principio de operación básico de ambas (SDXCentral, 2023).

En las redes de circuitos conmutados se debe crear una conexión física entre los dos nodos que quieren comunicarse, y esta conexión física debe mantenerse abierta durante todo el tiempo que se desee mantener la conexión abierta. Esto imposibilita que otro nodo pueda utilizar las secciones de circuitos que otro grupo de nodos está usando, además de imposibilitar la recepción de información de dos o más nodos de forma simultánea por el mismo medio.

En las redes de paquetes conmutados se divide la información que se quiere intercambiar en paquetes, que son enviados a través de la red de forma independiente. Cuando un nodo recibe un paquete cuyo destino no es él mismo,

reenvía el paquete hacia el siguiente nodo. De esta forma, se genera una malla de nodos interconectados que intercambian paquetes entre ellos, permitiendo así que varios nodos envíen y reciban paquetes sin tener que reservar una conexión directa y exclusiva entre ambos.

3.2 Ethernet

En 1973, mientras las ideas de DARPA se acercaban lentamente a los estándares actuales de Internet, nacía, de la mano de los ingenieros Bob Metcalfe y David Reeves Boggs, la primera versión de Ethernet en el Centro de Investigación de Palo Alto (PARC), propiedad de XEROX (Ullah, 2012). Esta versión es rápidamente reemplazada por Ethernet II, cuando Bob Metcalfe decide fundar 3Com y consigue que Intel y DEC decidan apostar por su tecnología. A partir de este punto, 3Com se centra en reunir y redactar toda la documentación necesaria para convertir a Ethernet en un estándar reconocido. En 1983 se crea la normativa IEEE 802.3, en la que se detallan las características de Ethernet y se convierte en otro estándar más para la conexión de equipos.

Durante la década de 1980 y 1990, Ethernet competirá con otros estándares, como Token-Ring, en lo que se conoce como las Guerras de los Protocolos.

3.3 IPX

Mientras Ethernet iba madurando en manos de Metcalfe y Reeves, un equipo del mismo centro de investigación de XEROX estaba creando una arquitectura de redes locales intercomunicadas en la que se pretendía usar PARC Universal Packet (PUP), una de las primeras suites de protocolos de red.
Novell utiliza como base para su suite de protocolos toda la investigación realizada por los ingenieros de XEROX PARC y en 1983 lanza NetWare, un sistema operativo de red que utiliza la pila de protocolos IPX/SPX (Novell, 2000).

Gracias a la popularidad de Novell NetWare, IPX se torna en uno de los grandes estándares para la interconexión de redes.

3.4 IP

La pila de protocolos TCP/IP nace como un sustitutivo al protocolo NCP, desarrollado por el Departamento de Defensa de los Estados Unidos. NCP era un protocolo extremadamente sencillo que se utilizaba en los inicios de ARPANET para realizar comunicaciones entre las máquinas de la red. Con el crecimiento de esta, los investigadores se dieron cuenta de que necesitaban una suite de protocolos más robusta que la que tenían. De esta necesidad surge TCP/IP en 1981 (Kessler, 2004).

El auge de TCP/IP surge de la inclusión por defecto de su pila tecnológica en UNIX, lo que fue desbancando lentamente a sus competidores, que requerían de la instalación de software y drivers adicionales.

Con la inclusión de la posibilidad de utilizar TCP/IP en los productos de Novell, y la apuesta de IBM y otras grandes empresas tecnológicas por la suite de protocolos IP, TCP/IP se convierte en el estándar de facto para Internet y la mayoría de las redes; desbancando así a IPX/SPX.

Capítulo IV: Marco Teórico

En el capítulo anterior se repasa de forma breve y superficial la historia de parte de las tecnologías más importantes en el ámbito de las redes e internet, pero una visión puramente histórica no es suficiente para entender completamente el alcance de este proyecto.

Este capítulo está dividido en dos grandes secciones. La primera sección busca profundizar en los elementos necesarios para entender el diseño de un protocolo de red a nivel conceptual; mientras que la segunda se centra en los aspectos más importantes de la implementación y las tecnologías utilizadas. Aunque es cierto que en diversas ocasiones la línea que separa el diseño conceptual y la aplicación de este diseño al software y al hardware es muy difusa.

4.1 Networking

Como se ha descrito anteriormente, esta sección está dedicada a la explicación de los diferentes elementos necesarios para entender el diseño del producto.

4.1.1 Packet Switching

Como se ha explicado anteriormente, las redes de paquetes conmutados nacen en contraposición de las redes de circuitos conmutados. La principal diferencia entre ambas es la capacidad de poder utilizar todos los enlaces de cualquier nodo de la red en todo momento, ya que ningún enlace es usado de forma exclusiva.

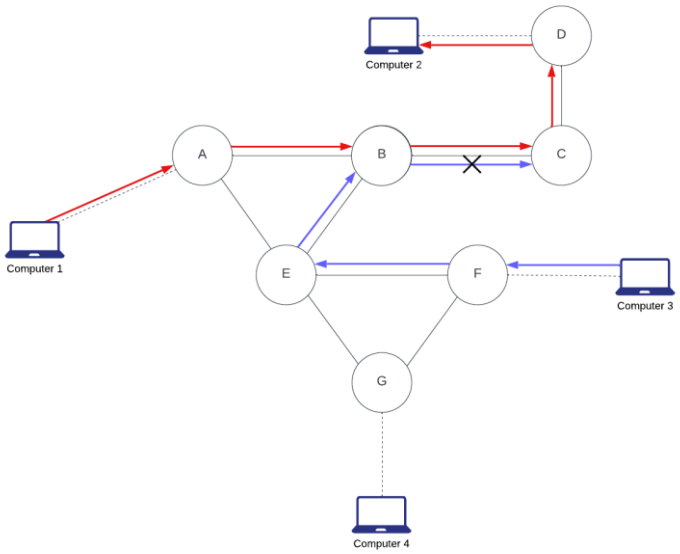


Figura 1 Diagrama de un circuito conmutado. Fuente: propia.

En la figura anterior se puede observar cómo, en una red de circuitos conmutados, si el ordenador Computer 1 ha establecido comunicación con Computer 2 entonces es imposible para cualquier otro ordenador de la red, en este caso Computer 3, el utilizar los nodos en uso por la conexión establecida. Esto impide que un dispositivo pueda mantener dos líneas de comunicación abiertas, a menos que se disponga de redundancia en el hardware, en cuyo caso el límite de conexiones simultáneas será determinado directamente por el número de redundancias de la red.

Cuando el concepto de conmutación de paquetes fue introducido, esta necesidad de reservar el hardware para establecer una conexión desapareció por completo. El sistema de Packet switching establece que cualquier información que se quiera enviar a través de una red debe ser encapsulada en paquetes. Estos paquetes son enviados a una red de nodos interconectados que decidirán, para cada paquete, qué camino debe seguir y tratarán de hacer el mayor esfuerzo posible para entregarlo a su destinatario.

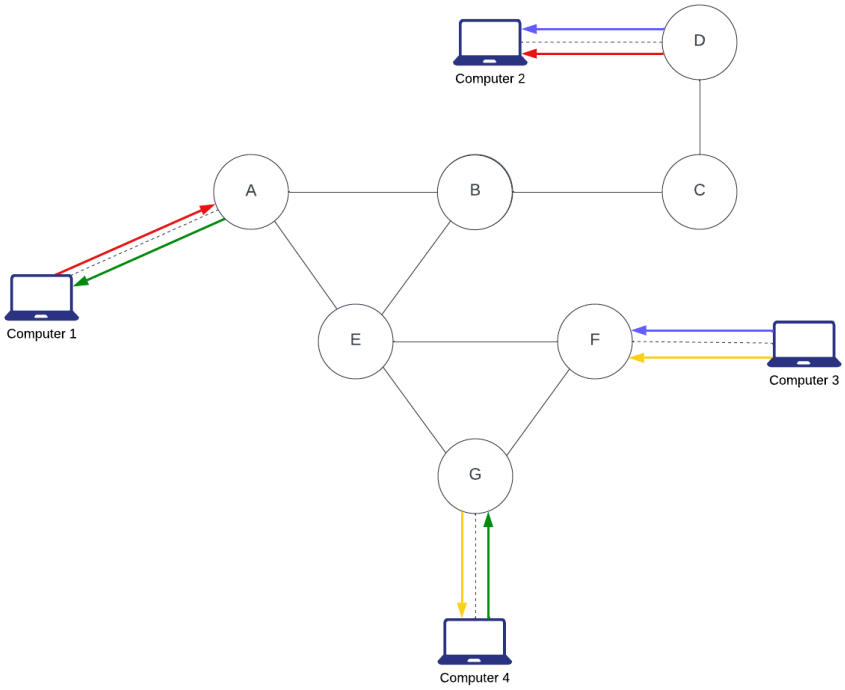


Figura 2 Diagrama de una red de paquetes conmutados. Fuente: propia.

Esto permite que la red de nodos se convierta en una caja negra, en la que lo importante son las entradas y las salidas del sistema y no el camino que los propios paquetes sigan. Esto, a su vez, presenta otros problemas que las redes de circuitos conmutados no tendrían, como puede ser una mayor pérdida de información, mayor latencia entre la emisión y la recepción de la información, la posibilidad de una saturación de la red, etc.

Sin embargo, estas desventajas son superadas en importancia por la flexibilidad de las redes de paquetes conmutados, su relativo bajo coste, un mayor uso de los elementos de la red y una mayor flexibilidad; además de la posibilidad de emular el comportamiento de las redes de circuitos conmutados mediante el uso de protocolos de capas superiores.

4.1.2 Modelo OSI

El modelo OSI se convierte en el estándar de facto para representar las comunicaciones de sistemas distribuidos en 1979 (Alani, 2014). Este modelo se basa

en una estructura de capas para dividir las diferentes funciones y responsabilidades que se requiere de la comunicación en un sistema distribuido. En concreto, el modelo OSI consiste en siete capas apiladas una encima de la otra: capa física (capa uno), capa de enlace de datos (capa dos), capa de red (capa tres), capa de transporte (capa cuatro), capa de sesión (capa cinco), capa de presentación (capa seis), y capa de aplicación (capa siete).

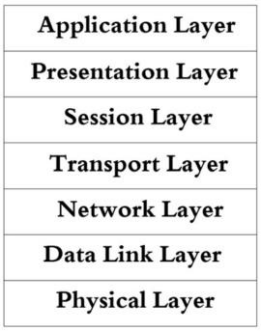


Figura 3 Pila del modelo OSI. Fuente: (Alani, 2014).

4.1.2.1 Capa física

La capa física del modelo OSI se encarga de la transmisión de los datos como bits a través de un medio. Es la responsable de que los bits transmitidos son recibidos de forma secuencial y ordenada.

Si se usara como ejemplo lo descrito por el Request For Comments 1149, la capa física sería la paloma mensajera y el medio en el que los bits son escritos (Waitzman, 1990).

En el caso de la transmisión a través de un medio más convencional, como un cable Ethernet, el estándar de transmisión de información (por ejemplo, 10BASE-T) formaría parte de la capa física.

4.1.2.2 Capa de enlace de datos

La capa de enlace de datos del modelo OSI ve la información a transmitir como tramas. Estas tramas tienen una cabecera y una cola que añaden información de control a los datos a transmitir.

Esta es una de las capas más complejas, ya que tiene un gran número de responsabilidades para garantizar el envío de datos a través de protocolos que requieren (o no) establecer conexión.

Un ejemplo de esta capa de enlace de datos es el protocolo Ethernet o el estándar 802.11 (WiFi).

4.1.2.3 Capa de red

La tercera capa del modelo OSI es la capa de red. En esta capa, la información a transmitir es interpretada como un paquete. Esta es la capa cuyo principal cometido es el enrutamiento de estos paquetes entre nodos, para permitir que un paquete pueda atravesar la red y llegar a su destino.

El enrutamiento se consigue mediante la asignación de direcciones (de capa 3) a los dispositivos de la red. Mediante estas direcciones, se puede escoger un camino para cada paquete que cruza la red y el destinatario puede recibir el paquete.

Además, esta capa debe ser capaz de fragmentar o segmentar los paquetes, permitiendo así que cruce redes con diferentes tamaños máximos de paquetes.

Uno ejemplos de protocolos de capa de red pueden ser Internet Protocol (IP) o Internetwork Packet Exchange (IPX).

4.1.2.4 Capa de transporte

La capa de transporte trata la información a transmitir como segmentos. Es la responsabilidad de esta capa el proveer, si es necesario, de soporte para flujos de datos orientados a conexión (permitiendo emular, de alguna forma, el funcionamiento de las redes de circuitos conmutados sin sus principales desventajas), garantizar la recepción y la correcta secuenciación de los datos enviados, y permitir tener control del flujo de segmentos.

Algunos ejemplos de protocolos de esta capa son Transmission Control Protocol (TCP), User Datagram Protocol (UDP), o Sequenced Packet Exchange (SPX).

4.1.2.5 Capa de sesión

La quinta capa del modelo OSI es la capa de sesión. Esta capa se encarga de establecer y finalizar sesiones, de controlar y almacenar los tokens para los servicios de autentificación y autorización, y de permitir restaurar y sincronizar estados en las sesiones.

Un ejemplo de protocolo de esta capa es Password Authentication Protocol (PAP).

4.1.2.6 Capa de presentación

La sexta capa del modelo OSI es la capa de presentación. Esta capa es responsable de como los datos son presentados a la aplicación, ofreciendo además servicios de compresión, encriptación, y traducción entre otros.

Un ejemplo de protocolo de esta capa es Secure Sockets Layer (SSL).

4.1.2.7 Capa de aplicación

La séptima y última capa del modelo OSI es la capa de aplicación. Esta es la capa con la que el usuario interactúa y es la encargada de transmitir los datos que el usuario desea enviar.

Un ejemplo de protocolo de capa siete es HyperText Transfer Protocol (HTTP).

4.1.3 Ethernet

En el apartado anterior se ha mencionado el estándar Ethernet; este estándar es un protocolo de capa dos del modelo OSI.

Ethernet tiene dos modos de funcionamiento. El primero es Half-Duplex, modo que permite compartir un medio para transmitir información; es decir, que permite más de dos dispositivos por dominio de colisión. Debido a esto, es posible que se produzcan colisiones de tramas, y por eso se requiere de un mecanismo para detectar colisiones. Este mecanismo es conocido como Carrier Sense with Multiple Access and Collision Detection (CSMA/CD) (Spurgeon, 2000). Además de depender de CSMA/CD para funcionar correctamente, también es necesario tener en cuenta la longitud total del dominio de colisión; ya que, si excede el máximo soportado por el estándar, CSMA/CD deja de funcionar.

El segundo modo es Full-Duplex, modo que se supone como estándar en este proyecto. Este modo desactiva algunas características, como CSMA/CD, y permite solo dos dispositivos por dominio de colisión. Esto permite, además de descartar los límites de distancia de CSMA/CD, operar transmitiendo y recibiendo datos al mismo tiempo.

4.1.3.1 Estructura de una trama Ethernet

A pesar de que normalmente se toma el tipo de trama Ethernet II como el único estándar, hubo una época en que más de un tipo de trama estaban definidas y en uso (Provan, 1993).

Estos tipos de trama tienen unos elementos comunes, como son las direcciones físicas (MAC) de origen y de destino, pero varían en el resto de los campos.

4.1.3.1.1 Ethernet II

La trama Ethernet II está compuesta por tres campos. Los dos primeros son las direcciones MAC de destino y origen, que ocupan seis octetos cada una. El tercer campo es el EtherType, que ocupa dos bytes de la trama. Este campo aporta información acerca del protocolo de capa superior que está transportando la trama. Los EtherTypes más comunes son los siguientes: 0x0800 (IP), 0x0806 (ARP), 0x8137 o 0x8138 (IPX) (Eastlake & Zuniga, 2022).

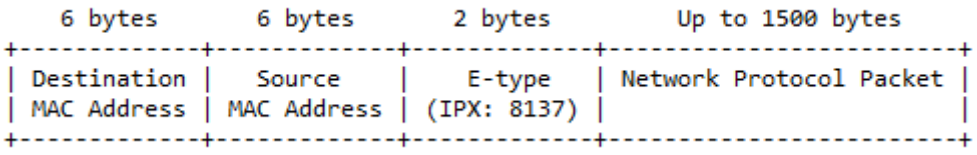


Figura 4 Formato de una trama Ethernet II. Fuente: (Provan, 1993).

4.1.3.1.2 Raw 802.3

La trama Raw 802.3 está compuesta por tres campos.
Los dos primeros campos son las direcciones MAC de destino y origen, que ocupan los primeros doce octetos en total.
El tercer campo, que ocupa dos octetos de la trama, indica el tamaño total del paquete. Esta trama no dispone de un mecanismo para identificar el tipo de tráfico que está transportando, por lo que depende de que solo se esté transportando un tipo de tráfico o requiere que los dos primeros bytes del tráfico a enviar indiquen el tipo de protocolo de capa superior.

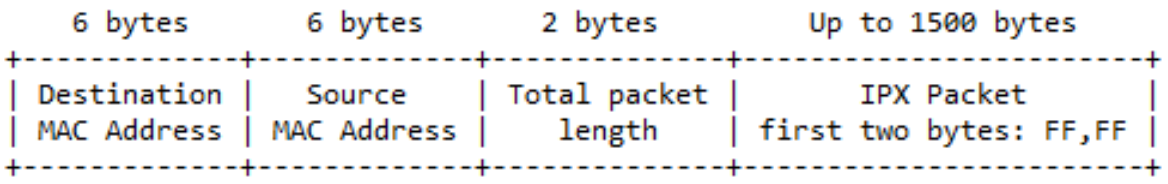


Figura 5 Formato de una trama Raw 802.3. Fuente: (Provan, 1993).

4.1.3.1.3 802.3/802.2

La trama 802.3/802.2 está compuesta por los tres campos de la trama raw 802.3 más la cabecera LLC (IEEE International Standard for Information technology, 1998). Esta cabecera cuenta con tres campos.

El primer campo es Destination Service Access Point (DSAP), que ocupa un octeto; el segundo campo es Source Service Access Point (SSAP), que ocupa un byte; y el tercero es el campo de control, que normalmente ocupa un byte, pero que puede llegar a ocupar dieciséis bits en caso de que se le añada un número de secuencia.

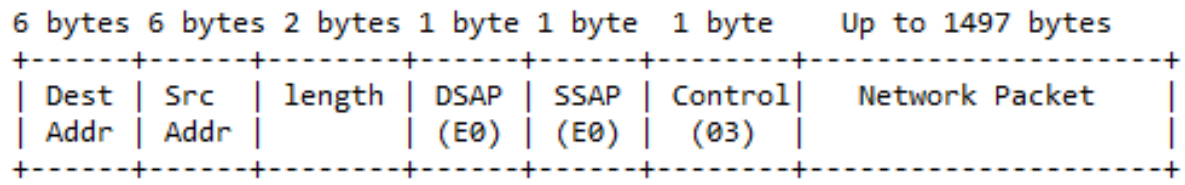


Figura 6 Formato de una trama 802.3. Fuente: (Provan, 1993).

4.1.3.1.4 802.3/SNAP

La trama 802.3/SNAP está compuesta por los campos de la trama 802.3/802.2 más la cabecera SNAP. Esta cabecera, conocida como SNAP ID, incluye tres bytes que indican el ID de protocolo, y otros dos octetos que transportan el EtherType de la trama Ethernet II (Postel & Reynolds, A Standard for the Transmission of IP Datagrams over IEEE 802 Networks, 1988).

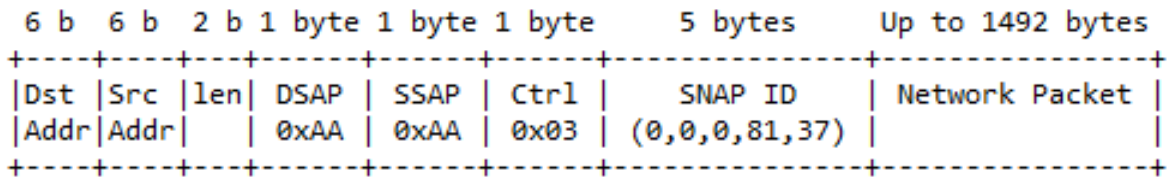


Figura 7 Formato de una trama 802.3/SNAP. Fuente: (Provan, 1993).

4.1.3.2 Media Access Control (MAC)

En el apartado anterior se ha mencionado el término dirección MAC. Este término hace referencia a una subcapa de la capa dos del modelo OSI para todas las comunicaciones basadas en el estándar IEEE 802.

En este estándar se define una dirección MAC como un tipo de dirección de cuarenta y ocho bits en los que los primeros dos bits indican si el identificador es globalmente único o si está administrado de forma local y si se trata de una dirección unicast o multicast, los siguientes veintidós bits identifican a la organización a la que pertenece ese dispositivo, y los otros veinticuatro identifican a la tarjeta de red (IEEE International Standard for Information Technology, 2002).

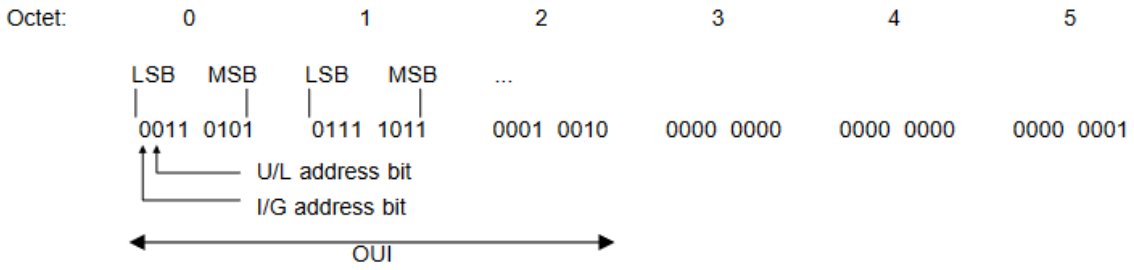


Figura 8 Formato de una dirección MAC. Fuente: (IEEE International Standard for Information Technology, 2002).

Estas direcciones deben ser únicas para un mismo dominio de broadcast de capa dos, para que la red funcione correctamente.

4.1.4 Unicast, Broadcast y Multicast

En la sección anterior se ha escrito sobre direcciones multicast, unicast, y broadcast (Kaur, Singh, & Singh, 2016). Estos tres términos se refieren a los tres esquemas de direccionamiento que se pueden utilizar en una red.

Cuando se usa el esquema unicast, la dirección está identificando a un único dispositivo; por lo que los dispositivos de la capa solo deben hacer llegar el paquete o la trama a ese dispositivo (en la mayoría de los casos).

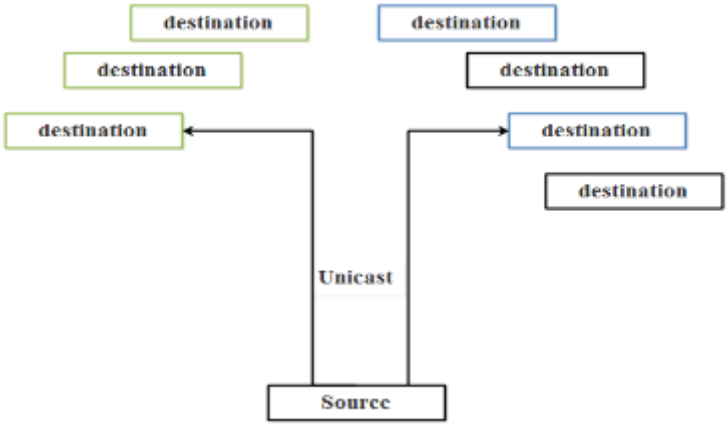


Figura 9 Transmisión de una trama unicast. Fuente: (Kaur, Singh, & Singh, 2016).

Cuando se usa el esquema broadcast, la dirección está identificando a todos los posibles dispositivos de ese dominio de broadcast; por lo que los dispositivos se encargan de transmitir esa trama o paquete a todos los dispositivos.

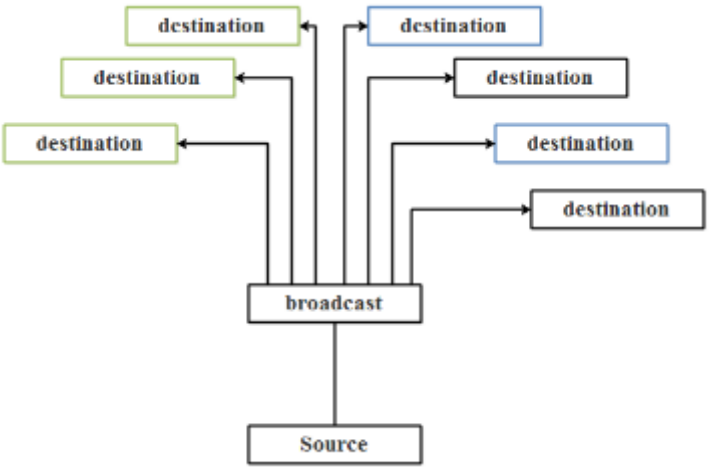


Figura 10 Transmisión de una trama broadcast. Fuente: (Kaur, Singh, & Singh, 2016).

Cuando se usa el esquema multicast, la dirección está identificando a los usuarios suscritos a un grupo multicast. Los dispositivos de la red saben cuándo transmitir, o no, la trama en función de si el usuario está o no suscrito al grupo destino.

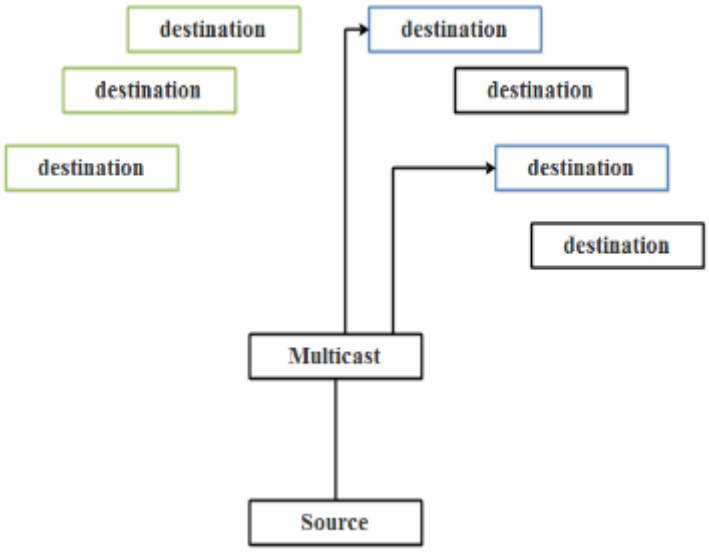


Figura 11 Transmisión de una trama multicast. Fuente: (Kaur, Singh, & Singh, 2016).

Mediante el uso correcto de estos esquemas de direccionamiento es posible mejorar la efectividad de la red y evitar la saturación.

4.1.5 Internetwork Packet Exchange

El protocolo de red IPX fue creado por Novell basándose en el trabajo realizado por XEROX en XNS e IDP (Novell, 2001).

Este protocolo fue uno de los protocolos más usados en las redes corporativas debido al éxito de Novell NetWare, un sistema operativo de red que funcionaba sobre IPX.

Este protocolo de capa tres tiene la siguiente estructura: una cabecera de treinta bytes del propio protocolo, y una sección de datos, que normalmente incluye otras cabeceras de protocolos superiores.

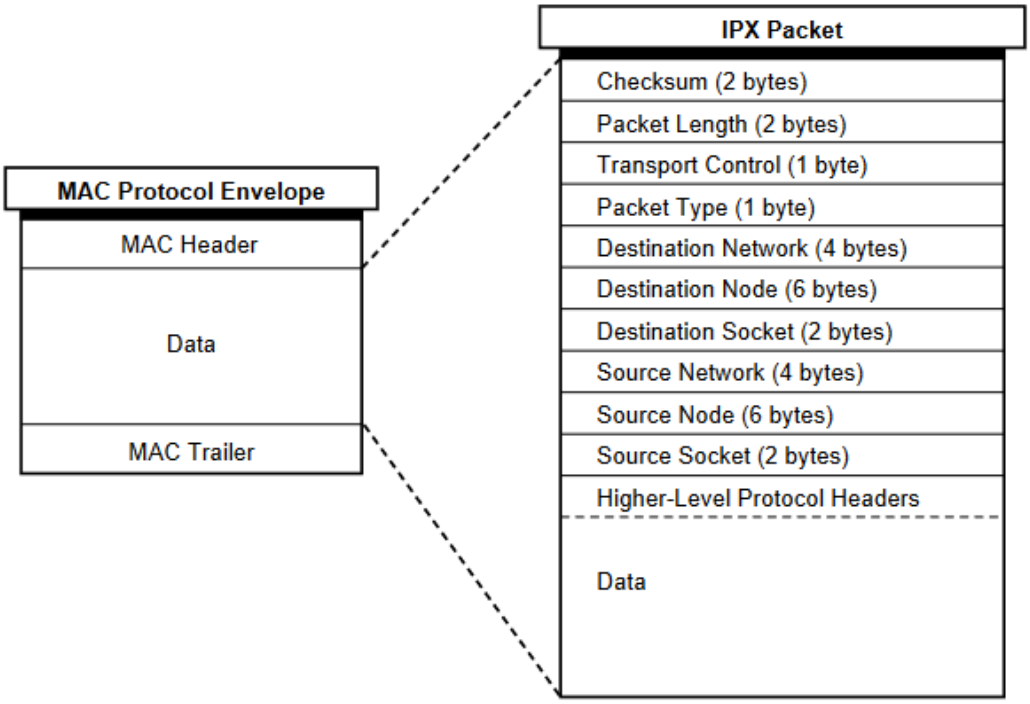


Figura 12 Estructura de un paquete IPX. Fuente: (Novell, 2001) .

4.1.5.1 Checksum

Estos dos primeros octetos del paquete se usaban, de forma tradicional, para indicar que el paquete era un paquete IPX en redes raw 802.3 (Provan, 1993). En caso de

estar funcionando en cualquier otro tipo de red, estos dos bytes se convierten en un checksum del paquete para poder verificar la integridad de este.

4.1.5.2 Packet Length

Los dos siguientes bytes del paquete indican el tamaño total del paquete IPX. Esto incluye tanto la cabecera como el resto de los datos.

4.1.5.3 Transport Control

El siguiente octeto del paquete indica la cantidad de saltos que el paquete ha dado hasta su ubicación actual. Por cada router que pasa el paquete, este valor es incrementado (a diferencia que TTL en los paquetes IP, que decrementa).

4.1.5.4 Packet Type

En el siguiente octeto del paquete se indica el tipo de servicio al que pertenece el paquete. En función del valor, puede indicarse que es un paquete de tipo RIP (0x01), SPX (0x05), u otros.

4.1.5.5 Destination Network

Los siguientes cuatro bytes contienen la dirección de la red de destino. Si este campo contiene solo ceros significa que la red de destino es la actual.

4.1.5.6 Destination Node

Los siguientes seis bytes indican la dirección física (MAC) del nodo destino. En caso de que la dirección sea una dirección broadcast, el router destino se encargará de realizar el broadcast a toda la red destino.

4.1.5.7 Destination Socket

Los dos siguientes octetos están reservados para el socket destino. Este es el equivalente a un puerto en la pila TCP/IP. Los sockets se encargan de enrutar el tráfico dentro de una misma máquina, asignando un socket a cada proceso que quiera enviar o recibir información mediante IPX.

4.1.5.8 Source Network

Los siguientes cuatro bytes contienen la dirección de la red de origen. Si este campo contiene solo ceros significa que la red de origen es desconocida. A pesar de que un paquete con solo ceros en la dirección destino no puede atravesar un router IPX, un paquete con la dirección de origen de esta misma forma sí que puede atravesarlo.

4.1.5.9 Source Node

Los siguientes seis bytes indican la dirección física (MAC) del nodo origen. Las direcciones broadcast no son una dirección de origen válida.

4.1.5.10 Source Socket

Los dos siguientes octetos están reservados para el socket origen.

4.1.5.11 Higher Level Protocol Headers y Data

A partir de este punto del paquete, IPX trata el resto de los bits como datos a transmitir. Lo más común es que a continuación haya un encabezado del siguiente protocolo, pero es posible transmitir datos directamente sobre IPX sin necesidad de tener un protocolo superior.

4.1.6 Internet Protocol version 4

El protocolo IPv4 es la versión más extendida de IP en la actualidad. La cabecera de este protocolo, en su mínima expresión, consiste en veinte bytes (Information Sciences Institute University of Southern California, 1981).

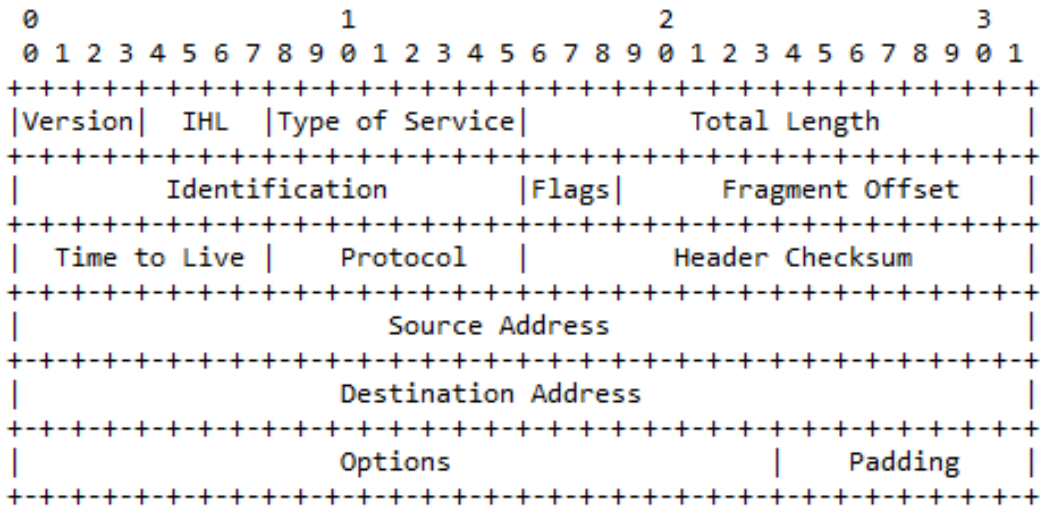


Figura 13 Diagrama de una cabecera IPv4. Fuente: (Information Sciences Institute University of Southern California, 1981).

4.1.6.1 Version

Los primeros cuatro bits de la cabecera indican la versión de la especificación del protocolo. En el caso de la versión cuatro, este campo debe ser 4.

4.1.6.2 Internet Header Length

Los siguientes cuatro bits indican el número de conjuntos de 32 bits que ocupa la cabecera IP. Como el tamaño mínimo de esta es de 20 bytes, el valor mínimo que debe tener este campo es 5.

4.1.6.3 Type of service

El tipo de servicio encapsula el siguiente byte la cabecera. En esta sección se indica el tipo de calidad de servicio que se desea para este paquete, aunque no se asegura en ningún momento que la calidad de servicio solicitada sea la recibida.

4.1.6.4 Total Length

Los siguientes dos octetos indican el tamaño total del paquete, incluyendo la cabecera y los datos.

4.1.6.5 Identificación

Estos dos bytes contienen un número que posibilita la ordenación de datagramas fragmentados.

4.1.6.6 Flags

Estos tres bits controlan diversos aspectos de la fragmentación de datagramas. El primer bit siempre tiene que ser cero, el segundo indica si se puede o no se puede fragmentar un datagrama, y el último indica si quedan más fragmentos del datagrama por llegar.

4.1.6.7 Fragment Offset

Estos trece bits indican qué parte del datagrama fragmentado contiene ese paquete. Se mide en grupos de 64 bits, siendo siempre el primer offset cero.

4.1.6.8 Time To Live

El siguiente octeto indica el tiempo de vida del paquete. Este valor es decrementado por cada router por el que pasa el paquete. Si este campo llega a cero, el router que ha decrementado el valor descarta inmediatamente el paquete.

4.1.6.9 Protocol

Este campo de un byte indica el protocolo de capa superior que está encapsulado en la sección de datos.

4.1.6.10 Header Checksum

Estos dieciséis bits contienen un checksum de la cabecera IP. Si la cabecera IP está dañada, el paquete se descarta automáticamente.

4.1.6.11 Source Address

Los siguientes cuatro octetos son la dirección IP de origen. Esta dirección identifica la máquina que envió el paquete. Esta dirección no tiene relación alguna con la dirección MAC, a diferencia de IPX.

4.1.6.12 Destination Address

Los siguientes cuatro octetos son la dirección IP de destino. Esta dirección identifica la máquina que debe recibir el paquete. Esta dirección no tiene relación alguna con la dirección MAC, a diferencia de IPX.

4.1.6.12 Options y Padding

Estos dos campos son opcionales, especialmente el padding, que se utiliza para asegurarse de que la sección de datos del paquete esté alineada a 32 bits.

4.1.7 Internet Protocol version 6

IPv6 es la versión más reciente del protocolo de Internet. El tamaño de la cabecera es de cuarenta bytes (Deering & Hinden, 2017).

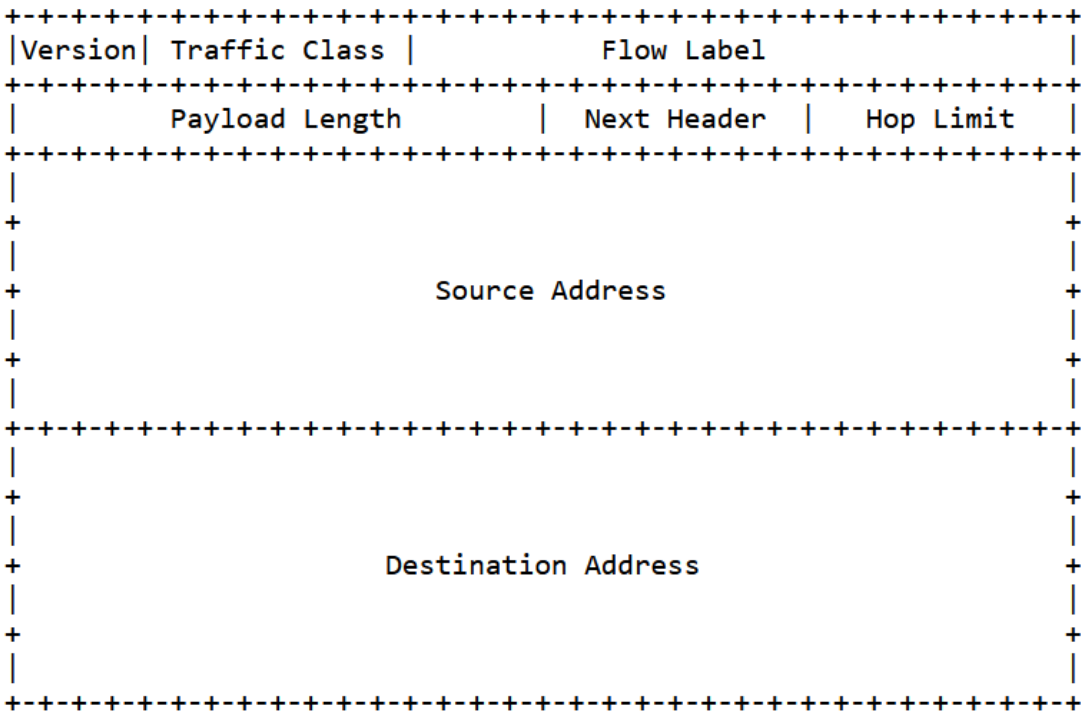


Figura 14 Diagrama de una cabecera IPv6. Fuente: (Deering & Hinden, 2017).

4.1.7.1 Version

Al igual que en la versión cuatro del protocolo, este campo de cuatro bits indica la versión de la cabecera. En esta versión el número debe ser seis.

4.1.7.2 Traffic Class

Este campo de ocho bits es utilizado para la gestión del tráfico de red y puede ser alterado en cualquier momento por cualquier dispositivo.

4.1.7.3 Flow Label

Este campo de veinte bits se usa para marcar los paquetes de un mismo flujo de datos (Amante, Carpenter, Jiang, & Rajahalme, 2011).

4.1.7.4 Payload Length

Este campo de dos bytes indica el tamaño de los datos encapsulados en el paquete sin contar la cabecera IP. Se indica en número de bytes.

4.1.7.5 Next Header

En este octeto se indica el tipo de cabecera del siguiente protocolo de nivel superior incluido en los datos a transmitir. Usa los mismos valores que el campo Protocol en la cabecera de versión cuatro.

4.1.7.6 Hop Limit

Este byte indica el tiempo de vida de un paquete del mismo modo que lo hace TTL en la cabecera de la cuarta versión del protocolo.

4.1.7.7 Source Address

Los siguientes 128 bits están dedicados a la dirección de origen del paquete IPv6. Como en la cabecera de la cuarta versión, este campo no tiene nada que ver con la dirección física (MAC).

4.1.7.7 Destination Address

Los siguientes 128 bits están dedicados a la dirección de destino del paquete IPv6. Como en la cabecera de la cuarta versión, este campo no tiene nada que ver con la dirección física (MAC).

4.1.8 Transmission Control Protocol

El protocolo TCP, a pesar de que ha sufrido diversas modificaciones desde su creación en 1981 (Information Sciences Institute University of Southern California,

1981), ha mantenido su cabecera prácticamente intacta, haciendo así que las nuevas especificaciones fueran compatibles con la original.

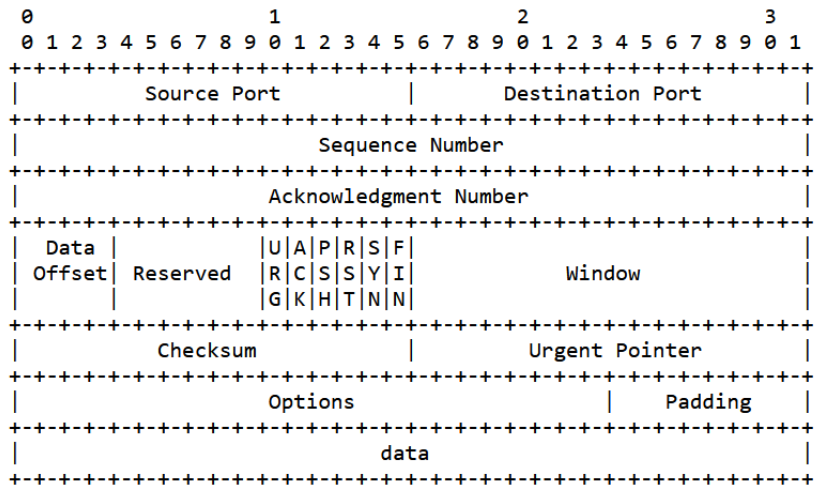


Figura 15 Diagrama de la especificación original de TCP. Fuente: (Information Sciences Institute University of Southern California, 1981).

La cabecera TCP contiene veinte bytes de información como mínimo, siendo acompañado de una sección de elementos opcionales que pueden agrandar la cabecera y tienen que estar alineados a ocho bits, posteriormente añadiendo el padding necesario para que el conjunto de opcionalidades esté alineado a treinta y dos bits.

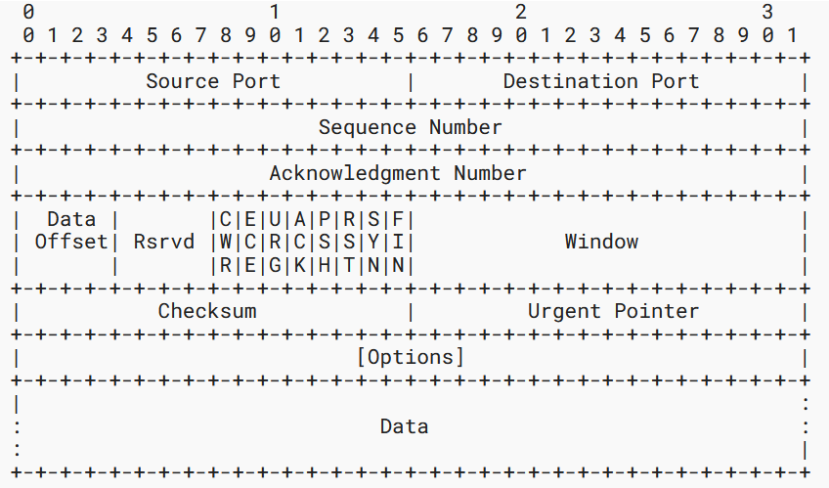


Figura 16 Diagrama de una cabecera TCP. Fuente: (W. Eddy, 2022).

4.1.8.1 Source Port

Este campo de dos bytes indica el puerto de origen.

4.1.8.2 Destination Port

El siguiente campo de dos bytes identifica el puerto de destino.

4.1.8.3 Sequence Number

El número de secuencia es el tercer campo de la cabecera TCP. Este campo de treinta y dos bits indica el número de secuencia del paquete. Si el flag SYN tiene un valor de uno, entonces el número de secuencia es el inicial.

4.1.8.4 Acknowledgement Number

El número de ACK indica, si el flag ACK tiene un valor de uno, el valor del número de secuencia que el emisor del paquete está esperando. Por este motivo, es un campo de cuatro bytes.

4.1.8.5 Data Offset

Este campo de cuatro bits indica el tamaño total de la cabecera TCP en grupos de treinta y dos bits. El valor mínimo de este campo es cinco, ya que el tamaño mínimo de la cabecera es de veinte bytes.

4.1.8.6 Reserved

Este campo, actualmente de tres bits, está reservado para un posible uso futuro.

4.1.8.7 Flags

Este campo, originalmente de seis bits y que actualmente ocupa nueve bits, es el único campo que ha cambiado en la cabecera TCP con el paso del tiempo. Esto se debe a que el campo Reserved antecede a este, permitiéndole así crecer en capacidad hasta ocupar el tamaño completo de su antecesor.

El campo tiene nueve valores de un bit, de entre los que se destacan el bit de ACK, el bit de SYN, el bit de FIN, y el bit de RST.

4.1.8.8 Window

Este campo de dos bytes indica la cantidad de paquetes que el emisor está dispuesto a recibir antes de recibir una respuesta de ACK por parte del receptor. Este número puede cambiar en el curso de una comunicación TCP.

4.1.8.9 Checksum

Los dos siguientes bytes corresponden al checksum, que indica al receptor si el paquete ha sufrido algún error de transmisión y está dañado. En caso de que lo esté, el receptor puede solicitar de nuevo el paquete.

4.1.8.10 Urgent Pointer

Este campo de dieciséis bits indica el valor del urgent pointer en caso de que el flag URG tenga un valor de uno. Este campo contiene el valor del número de secuencia más un offset.

4.1.9 User Datagram Protocol

El protocolo UDP es el protocolo de capa cuatro del modelo OSI más simple. Su cabecera solo ocupa ocho bytes y tiene cuatro campos. Este protocolo, a diferencia

de TCP, no establece ningún tipo de conexión ni garantiza la recepción ordenada de los paquetes.

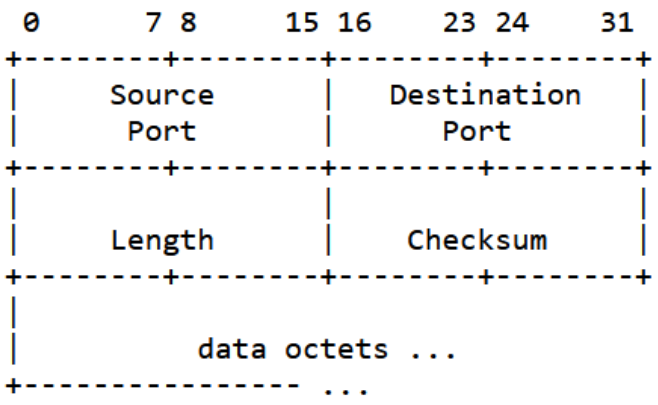


Figura 17 Diagrama de una cabecera UDP. Fuente: (Postel, User Datagram Protocol, 1980).

4.1.9.1 Source Port

Este campo de dos bytes indica el puerto de origen.

4.1.9.2 Destination Port

El siguiente campo de dos bytes identifica el puerto de destino.

4.1.9.3 Length

Este campo de dos bytes indica el tamaño total, en bytes, del datagrama (incluyendo la cabecera).

4.1.9.4 Checksum

Este campo de dos bytes indica al receptor si el paquete ha llegado correctamente o si ha sido corrompido en el transcurso del envío. En caso de que ese sea el caso, el paquete es ignorado.

4.2 Programación

Como se ha detallado anteriormente, esta sección del marco teórico está dedicada a describir los diferentes conceptos relativos a la programación y la implementación del proyecto.

4.2.1 Managed and Unmanaged languages

Un managed language, o managed code, es todo aquel código que está siendo gestionado por un runtime (Microsoft, 2021). Este término aparece, sobre todo, en la documentación relativa a .Net, donde el runtime en cuestión donde corre el código es la Common Language Runtime (CLR). Usualmente se entiende como managed languages no solo aquellos que corren sobre la CLR, sino todos aquellos lenguajes que cuentan con gestión de memoria automática, donde el programador no tiene que lidiar, normalmente, con asignar y liberar memoria.

Hay lenguajes de programación donde ambos paradigmas están disponibles, como puede ser C# (mediante el uso de unsafe) o C++/CLI (también conocido como Managed C++).

Capítulo V: Diseño metodológico i cronograma

En este capítulo se describen las dos principales metodologías usadas para realizar este proyecto de fin de grado.

5.1 Metodología de búsqueda de información

La metodología utilizada para la búsqueda de información ha sido la búsqueda de información mediante el uso de un motor de búsqueda. En esta fase se contemplan todas las posibles fuentes de información, sean, o no, fiables. La inclusión de fuentes no fiables para este trabajo de fin de grado es debido a la poca documentación oficial o de fuentes fiables que se puede obtener acerca de diversos aspectos técnicos de tecnologías ampliamente obsoletas hoy en día.

Una vez se ha obtenido esta recopilación de información; se compara la información de las diferentes fuentes y se identifican las discrepancias entre ellas, dando más fiabilidad a aquellas fuentes consideradas como fiables.

Una vez se tiene un panorama claro de qué información es fiable y qué información no lo es, se deciden las fuentes a tomar en cuenta en función de su lo completas que están.

Cuando se ha pasado por este proceso, se tiene un listado de fuentes fiables con información completa, y una serie de fuentes no fiables, que parecen coherentes, cuando no se ha podido encontrar información fiable sobre una tecnología en desuso.

Con este listado de fuentes, se logra crear un mapa conceptual claro de los conceptos y tecnologías a utilizar y sus respectivas definiciones.

5.2 Metodología de desarrollo

En el desarrollo de este proyecto se ha decidido utilizar una metodología clásica y bien asentada en procesos empresariales como es Waterfall. Se planteó utilizar metodologías ágiles, pero con el relativo poco tiempo para iterar y la necesidad de tener un diseño robusto y fuerte al que aferrarse la han convertido en una metodología que no se adecua al proyecto.

Se han definido dos fases para este desarrollo.

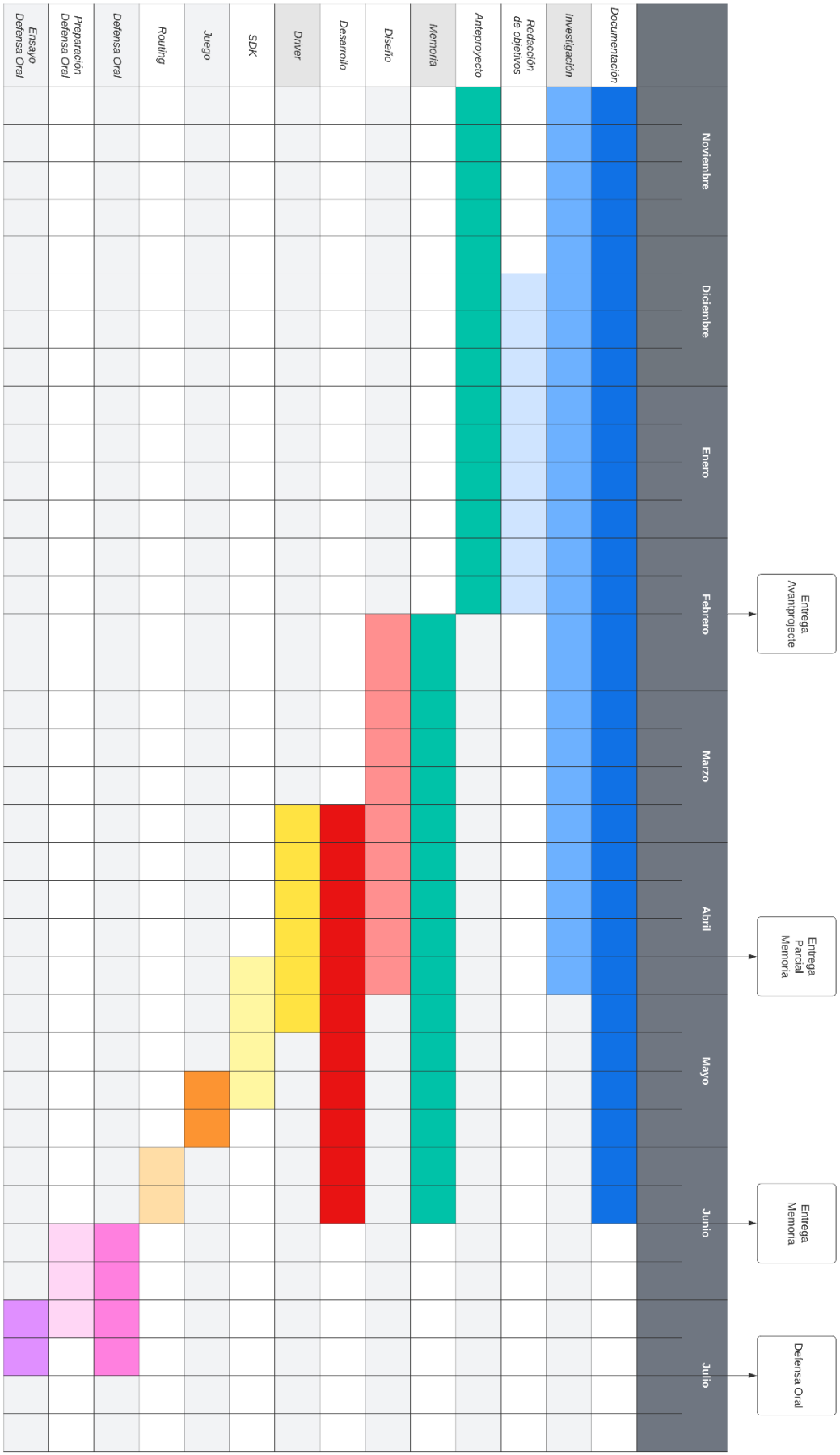
La primera fase consiste en una fase de diseño, en la que se pretende diseñar todos los protocolos necesarios para la pila de protocolos de red, incluyendo los protocolos de capa tres, cuatro, y siete necesarios. Con este diseño, y su respectiva documentación se procederá a la etapa de implementación.

En la etapa de implementación se plasmará el diseño realizado anteriormente en un software capaz de utilizar los protocolos, además de crear un SDK para poder implantarlo en otras aplicaciones. Por último, se desarrollará un entorno de pruebas en el que demostrar la viabilidad de esta nueva suite de protocolos.

Es posible que durante la fase de diseño se definan otros estándares no esenciales que no serán tenidos en cuenta durante la fase de implementación pero que sí forman parte de la especificación de los protocolos.

5.3 Cronograma

En esta sección se muestra el cronograma con los tiempos estimados para cada fase de la producción de este TFG.


Noviembre
Diciembre
Enero
Febrero
Marzo
Abril
Mayo
Junio
Julio
Documentación
Investigación
Redacción de objetivos
Anteproyecto
Memoria
Diseño
Desarrollo
Driver
SDK
Juego
Routing
Defensa Oral
Preparación Defensa Oral
Ensayo Defensa Oral
Entrega Avantprojecte
Entrega Parcial Memoria
Entrega Memoria
Defensa Oral

Capítulo VI: Diseño de la solución

El diseño de esta solución está separado en dos grandes bloques. El primero, el diseño de los protocolos consta de las especificaciones de los protocolos y su uso; el segundo, consta del diseño del software de la solución, donde se especifican los lenguajes de programación utilizados.

6.1 Diseño de los protocolos

En este apartado se definen seis protocolos de diferentes capas que son necesarios para el correcto funcionamiento de la suite de protocolos.

6.1.1 InterNetwork Packet Delivery Protocol (IPDP)

InterNetwork Packet Delivery Protocol (IPDP) es un protocolo de capa tres que forma el núcleo de esta suite de protocolos. Su función principal es la de permitir el enrutamiento de los paquetes, consiguiendo así que los paquetes puedan llegar de un punto P a un punto Q, pasando por una red de N nodos desconocidos. Por sí mismo este protocolo no garantiza la recepción de los paquetes ni su correcto orden de llegada.

Para el diseño de este protocolo se ha asumido que, las direcciones físicas de las máquinas origen y destino, no pueden superar los 64 bits. En caso de que estas redes hicieran uso de una tecnología con un direccionamiento mayor, se requeriría el uso de ARP y parte de las ventajas de esta suite desaparecerían.

En el caso de que la tecnología de capa dos sea Ethernet, la cabecera de la trama debe indicar 0x88B4 en el campo EtherType.

El diseño final de la cabecera de este protocolo consta de diversos campos alineados a 64 bits que suman en total 48 bytes. En caso de tener una MTU de 1500 bytes, esto

supone un 3,2% de cada paquete. En comparación, IPv6 tiene una cabecera de 40 bytes (un 2,7% de cada paquete). Si están habilitados los paquetes jumbo (paquetes con una MTU de 9000), estos porcentajes caen a 0,53% y 0,44%.

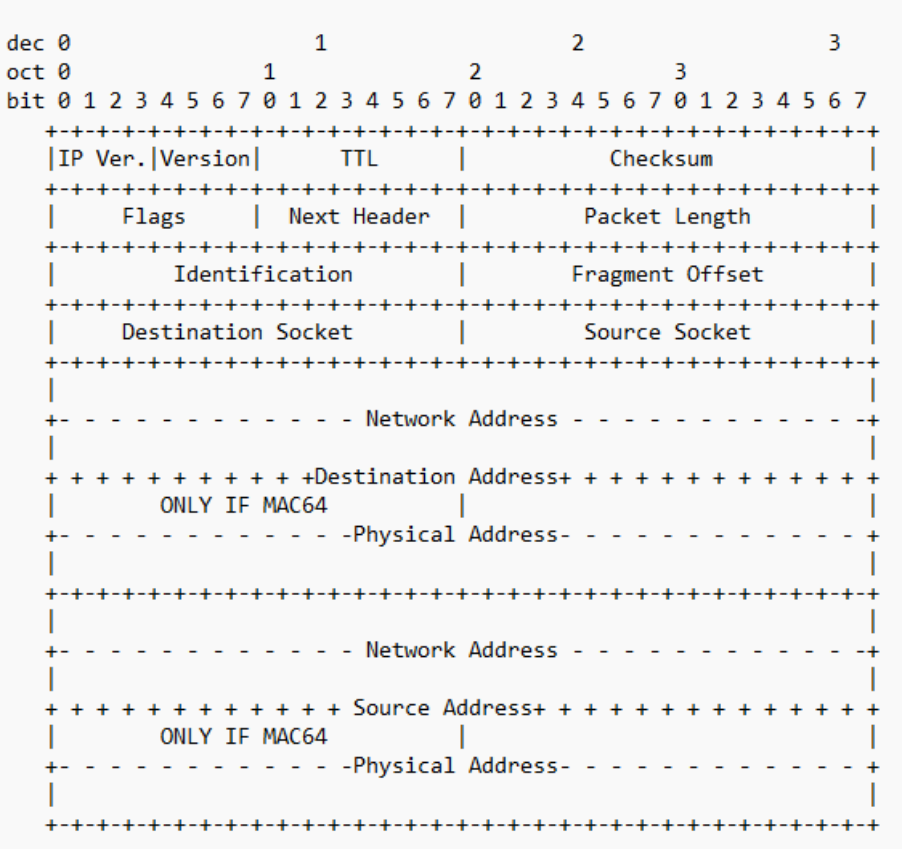


Figura 18 Cabecera IPDP. Fuente: propia.

La distribución general del paquete deja los primeros campos importantes alineados a 64 bits, permitiendo así realizar operaciones rápidamente y aunque el resto del paquete esté por llegar. La inclusión del campo TTL en estos primeros bytes permite descartar el paquete en caso de que el TTL sea 0.

La inclusión de algo parecido a los puertos de IP en esta cabecera permite a los firewalls realizar el trabajo del filtrado y bloqueo de paquetes de forma más sencilla, ya que no requieren analizar la cabecera de capa superior y todos los paquetes tienen socket origen y destino.

6.1.1.1 IP version + Version (8 bits):

Este campo indica la versión del protocolo IPDP. Por cuestiones de compatibilidad, los primeros cuatro bits nunca pueden ser ni 0100 (4) ni 0110 (6), porque están reservados para las versiones cuatro y seis de IP. Esta especificación dicta que, para este protocolo, los primeros cuatro bits deben ser 1001 seguidos de la versión del protocolo, que en su primera especificación es 0001.

6.1.1.2 TTL (8 bits):

Este campo indica el tiempo de vida del paquete. Durante el diseño de esta cabecera se planteó usar un tiempo de vida ascendente, permitiendo así saber cuántos saltos había dado un paquete antes de llegar a su destino, pero se desestimó debido a que de esta forma se filtraba información de la red. Por ende, este campo se comporta de la misma forma que su homólogo en IP: parte del host origen con un número determinado y este va disminuyendo cada vez que pasa por un dispositivo con capacidades de capa 3. Cuando este campo llega a cero, el paquete es descartado y se genera un paquete MCP (Message Control Protocol) indicando un error de Destino Inalcanzable.

6.1.1.3 Checksum (16 bits):

Este campo de 16 bits contiene el checksum del paquete. Para realizarlo, se hace uso de CRC-16 para calcular el checksum de todo el paquete, incluyendo la cabecera. Debido a que el campo Checksum no está poblado todavía se llena el campo de unos para este cálculo. Una vez se tiene el checksum, se sustituyen los unos por los dieciséis bits que correspondan.

En el host destino, se debe extraer el contenido del campo Checksum y compararlo con los bits obtenidos por CRC-16 de todo el paquete (teniendo en cuenta que el campo Checksum debe volver a ser poblado de dieciséis unos). Si el valor entre lo obtenido y el checksum del paquete difiere, este es descartado.

Este proceso es solo necesario en caso de que el flag CHK esté habilitado y, en caso contrario, el campo checksum estará, por defecto, poblado por dieciséis unos; no siendo necesario comprobar, en destino, si esta cadena de bits ha llegado correctamente.

6.1.1.4 Flags (8 bits):

Este campo contiene diversos marcadores, o flags, que aportan información al dispositivo receptor acerca de las características del paquete y de sus solicitudes.

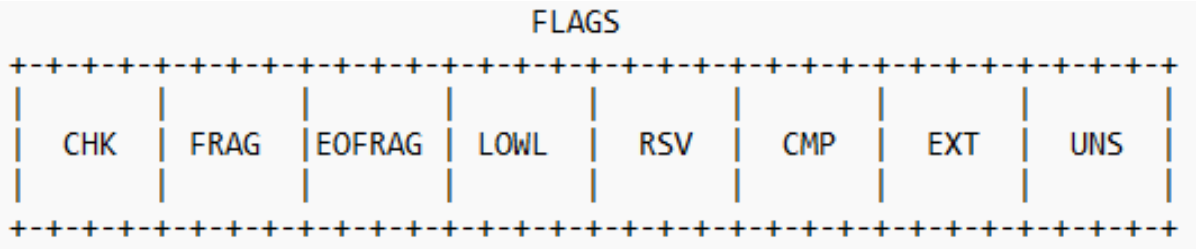


Figura 19 Flags de una cabecera IPDP. Fuente: propia.

Actualmente, solo hay siete flags definidos.

6.1.1.4.1 CHK

Cuando este bit está habilitado (poblado con un uno), se indica al dispositivo que el paquete ha sido sometido a un checksum y que el valor del campo checksum indica el valor obtenido por el cálculo del CRC-16.

6.1.1.4.2 FRAG

Cuando este bit está habilitado, se indica al dispositivo receptor que el paquete ha sido fraccionado.

6.1.1.4.3 EOFRAG

Cuando este bit está habilitado, se indica al dispositivo receptor que el este paquete contiene el último fragmento del paquete. Si este flag está habilitado, pero el marcador FRAG no lo está, entonces se indica que el paquete no se puede fraccionar, y que, en caso de que este sea más grande que el MTU de la red a la que se dispone a acceder, se debe descartar e informar al host origen mediante un paquete MCP.

6.1.1.4.4 LOWL

Cuando este marcador está habilitado, se está solicitando al dispositivo de enrutamiento un modo de baja latencia. Esta solicitud no tiene por qué ser aceptada.

6.1.1.4.5 RSV

Cuando este marcador está habilitado, se indica al dispositivo de enrutamiento que el paquete forma parte de un flujo reservado mediante el protocolo RRP. En caso de que el número de flujo no esté en la tabla RRP, el paquete se descarta y se informa al host origen mediante un paquete MCP.

6.1.1.4.6 CMP

Cuando este marcador está habilitado, se está solicitando al enrutador más próximo el modo de compatibilidad con redes que utilizan un protocolo distinto a IPDP. Esto se realiza en el dispositivo mediante una tabla de conversión en la que se guardan los datos de origen y destino; para que cuando un paquete de un protocolo distinto a IPDP llegue, el router sea capaz de revisar la tabla y construir un nuevo paquete con en el protocolo correcto.

Cuando un paquete IPDP utiliza esta tecnología, el flag EOFRAG debe estar siempre habilitado, ya que el campo Fragment Offset se utiliza para indicar el tipo de dirección destino.

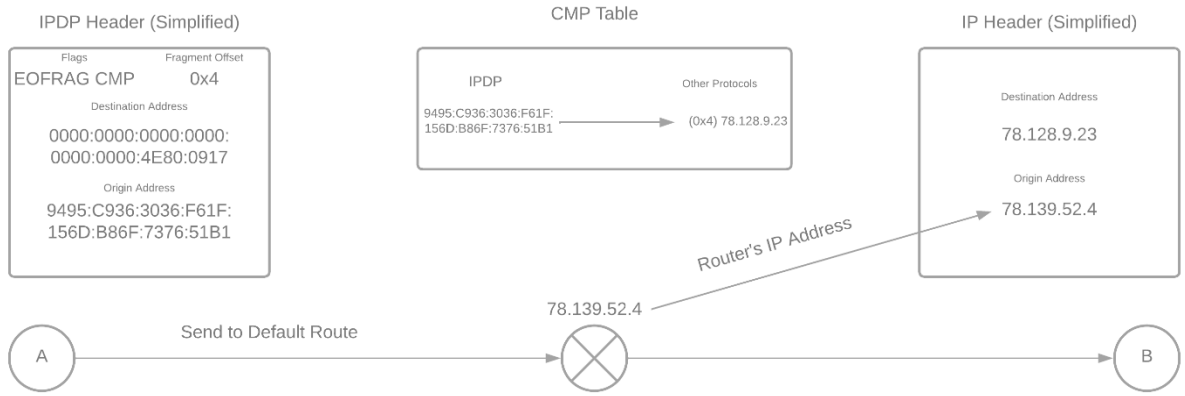


Figura 20 Diagrama del modo CMP de IPDP. Fuente: propia.

En el diagrama anterior se puede observar el funcionamiento del modo CMP. Un paquete es enviado hacia la ruta por defecto con la dirección destino en IPv4. Al pasar por el primer dispositivo de enrutamiento, este detecta el flag CMP y se dispone a convertir el paquete y a guardar en la tabla CMP las direcciones IP e IPDP.

En caso de que haya varios hosts que quieran establecer una conexión mediante el modo CMP a la misma dirección; requerirá que se establezca una identificación para cada flujo de información, ya sea mediante el uso de una cabecera extra o mediante el uso del campo Identification de la cabecera.

Este modo está diseñado como una solución para comunicarse con equipos antiguos que no puedan cambiar su pila de red, por lo que se espera que la aplicación del host destino sea capaz de utilizar el campo Identification para mantener separados los flujos de datos.

6.1.1.4.7 EXT

Cuando está activado este marcador, indica que se está operando en modo IPDP extendido. Esto implica que la parte de la dirección de red de una dirección IPDP pasa de 64 bits a 128, aumentando el tamaño total de la cabecera.

Su uso está desaconsejado hasta que el número total de direcciones de red disponibles sea bajo.

6.1.1.5 Next Header (8 bits):

Este campo de la cabecera IPDP indica el protocolo de capa cuatro transportado.

6.1.1.6 Packet Length (16 bits):

Este campo de la cabecera indica el tamaño total del paquete en bytes, por lo que el valor mínimo de este campo es 48 (el tamaño de la cabecera) y el máximo es 65536.

6.1.1.7 Identification (16 bits):

Este campo contiene el número de identificación de un paquete fraccionado, en caso de que el flag FRAG esté habilitado. Todos los fragmentos de un mismo paquete deben tener el mismo número de identificación.

En caso de que el marcador FRAG no esté habilitado, este número puede usarse por el protocolo RRP para indicar el identificador del flujo de datos.

6.1.1.8 Fragment Offset (16 bits):

Este campo contiene la información necesaria para que la máquina destino sea capaz de volver a ensamblar un paquete fraccionado. El número que aquí se indica es la posición respecto al inicio del paquete original (en bytes) del cargamento del paquete.

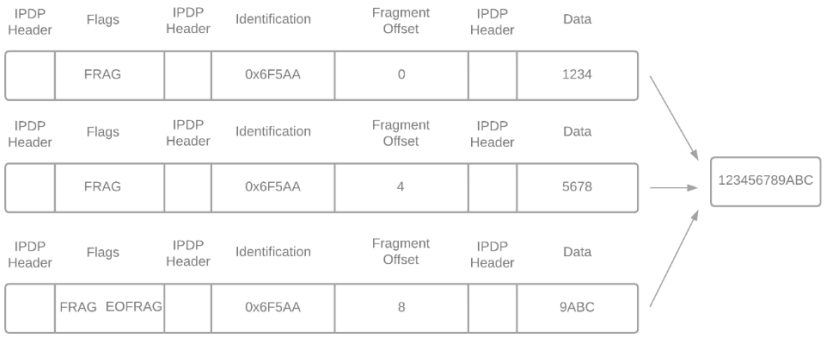


Figura 21 Reconstrucción de un paquete fraccionado. Fuente: propia.

6.1.1.9 Destination Socket (16 bits):

Este campo indica el socket destino. El socket es un número entre uno y 65536 que indica el puerto de la máquina destino al que va dirigido este paquete.

6.1.1.10 Source Socket (16 bits):

Este campo indica el socket origen. El socket es un número entre uno y 65536 que indica el puerto de la máquina origen desde el que el paquete ha salido (y al que se debe responder).

6.1.1.11 Destination Address (128 o 192 bits):

Este campo indica la dirección IPDP de destino. Una dirección IDPD está constituida de dos campos de 64 bits. El primero identifica la red de destino, y el segundo identifica la máquina física. Es debido a esto que este segundo campo debe contener la dirección física del adaptador. En Ethernet, como la dirección física es de 48 bits, los primeros dieciséis bits deben ser poblados con ceros.

En caso de que el flag EXT esté habilitado, la dirección IPDP pasa a componerse por 128 bits de dirección de red y 64 bits de dirección física; pasando a un tamaño de dirección de 192 bits.

6.1.1.12 Source Address (128 o 192 bits):

Este campo contiene la dirección IPDP de origen.

6.1.1.13 Fragmentación

Anteriormente se ha detallado el proceso de reconstrucción de un paquete fragmentado. En esta pequeña sección se muestra el proceso de fragmentación de IPDP.

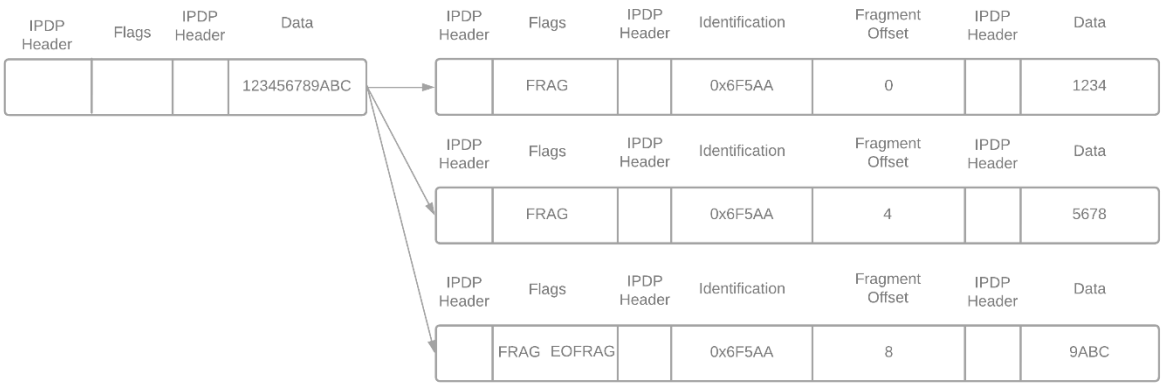


Figura 22 Fragmentación de un paquete IPDP. Fuente: propia.

Cuando un paquete IPDP pasa de una red con un MTU superior a otra con un MTU de menor tamaño, cabe la posibilidad de que el paquete deba ser fragmentado. Esto es debido a que la MTU indica el tamaño máximo de la carga que puede transportar la trama de capa dos.

Cuando esto sucede, el dispositivo de enrutamiento debe fragmentar el paquete. Si el paquete contiene un flag de EOFRAG y no contiene el marcador FRAG, se desestima el paquete y se informa al host origen que la MTU es inferior al tamaño del paquete. Esto es debido a que esta combinación de flags indica al dispositivo de enrutamiento que el paquete no se debe fragmentar (Don’t Fragment).

En caso de que ninguno de esos marcadores esté habilitado, comienza el proceso de fragmentación. En este proceso se genera un número de identificación, que permitirá al receptor poder reensamblar el paquete. Con este número se divide la carga en segmentos de inferior tamaño al MTU (teniendo en cuenta la cabecera IPDP). Una vez se han asignado el resto de los valores a las cabeceras de los nuevos paquetes,

se calcula el fragment offset, es decir, la posición en bytes respecto al inicio de la carga original.

Es importante destacar que un paquete no puede ser fragmentado más de una vez, en caso de que sea necesario volver a fragmentarlo se debe proceder como si se tratase de un paquete “Don’t Fragment”.

6.1.2 Packet Exchange Protocol (PEP)

Packet Exchange Protocol es el protocolo más simple de capa cuatro de la suite IPDP. Consta de tres campos, con un tamaño total de la cabecera de cuatro bytes.

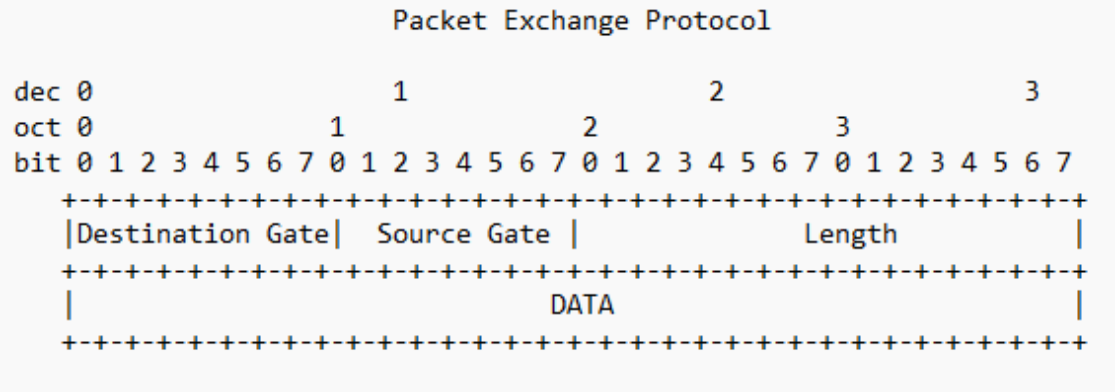


Figura 23 Diagrama de una cabecera PEP. Fuente: propia.

6.1.2.1 Destination Gate (8 bits)

Este campo de un byte indica la gate destino del paquete. Una gate es una especie de subpuerto en el que varias aplicaciones, que están dando un mismo servicio, pueden acoplarse. Esto permite la comunicación con varias instancias de una aplicación de forma más simple, a la vez que permite que más de una instancia esté bindeada a un mismo socket.

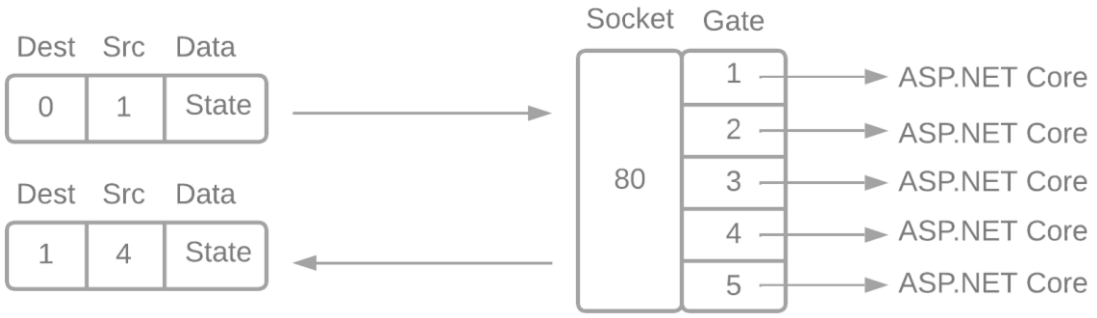


Figura 24 Diagrama de una petición mediante PEP. Fuente: propia.

El caso descrito en el diagrama muestra una petición (mediante PEP) a un servicio que conserva el estado de la sesión. Esto implica que, cuando el servicio responde, rellena el campo del gate origen con su número de gate; permitiendo así que el cliente se comunique directamente con él.

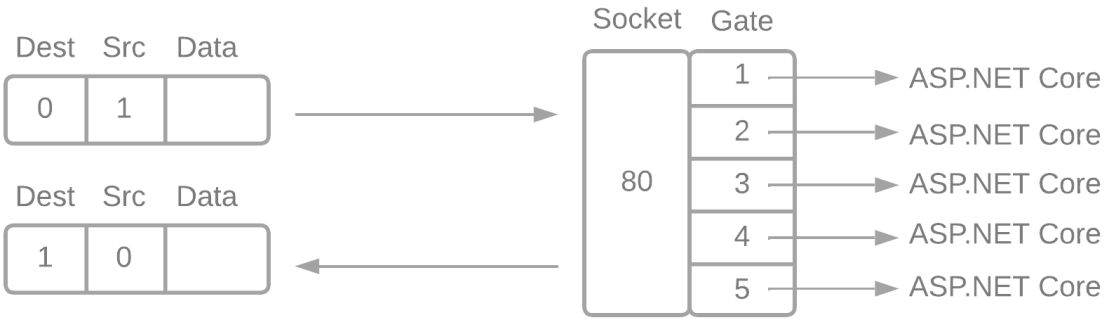


Figura 25 Diagrama de una petición stateless mediante PEP. Fuente: propia.

En caso de que el servicio no conserve el estado, y por ende no sea necesario que el cliente esté en comunicación siempre con la misma instancia, se puede asignar el valor de cero al gate para que la pila de red del sistema operativo decida a qué gate se deriva ese segmento.

6.1.2.2 Source Gate (8 bits)

Este campo indica el gate de origen. En caso de que la instancia de la aplicación no sea importante, este campo puede llenarse de ceros.

6.1.2.3 Length (16 bits)

Este campo indica el tamaño total del segmento en bytes, incluyendo la cabecera PEP.

6.1.3 Sequenced Packet Exchange Protocol (S-PEP)

Sequenced Packet Exchange Protocol, o Sequenced-PEP, es un protocolo orientado a conexión de capa cuatro de la suite de protocolos IPDP. Este protocolo se encarga del transporte de información entre hosts de forma confiable, permitiendo así que exista un flujo de información ordenada y sin errores, a costa de un menor rendimiento.

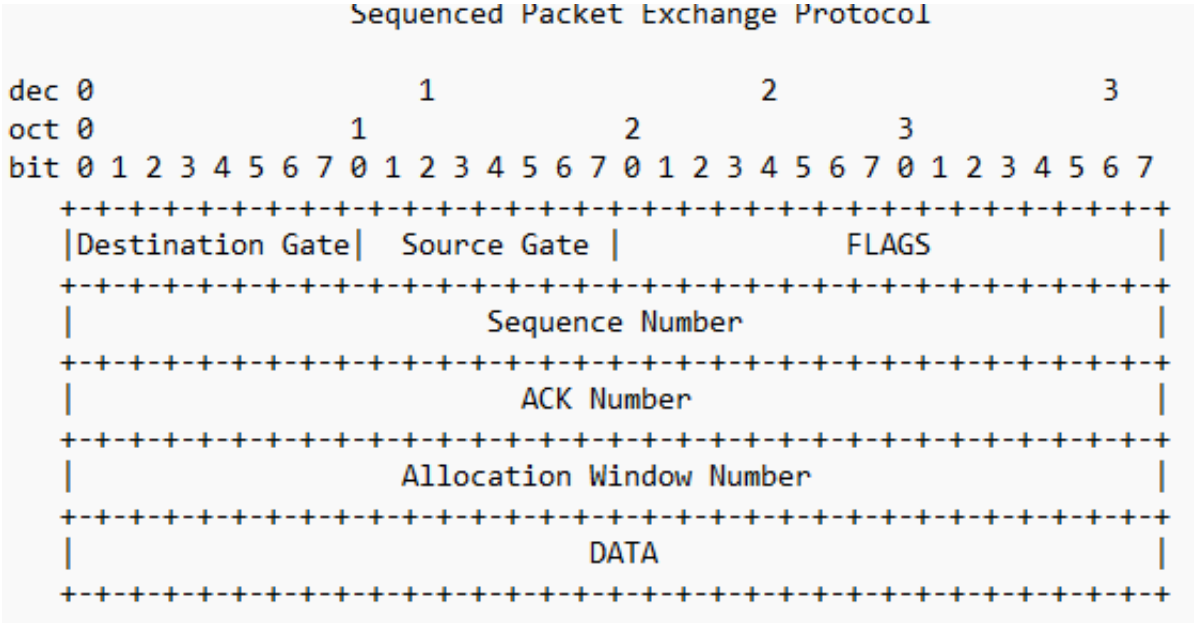


Figura 26 Diagrama de una cabecera S-PEP. Fuente: propia.

Este protocolo tiene una cabecera de dieciséis bytes con seis campos distintos.

6.1.3.1 Destination Gate (8 bits)

Como sucede con PEP, S-PEP también tiene el concepto de gates, por lo que este campo indica el gate destino del segmento.

6.1.3.2 Source Gate (8 bits)

Este campo contiene el gate origen del segmento.

6.1.3.3 FLAGS (16 bits)

Este campo contiene diversos marcadores, o flags, que aportan información al dispositivo receptor acerca de las características del segmento y de sus solicitudes.

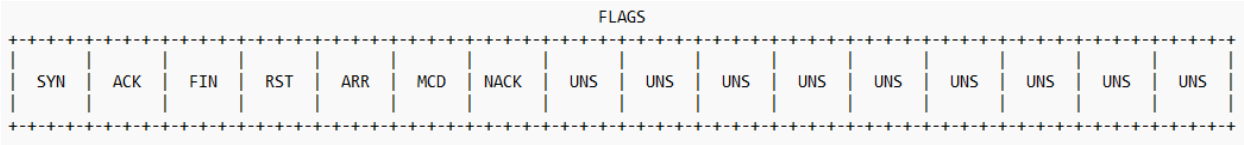


Figura 27 Diagrama de los flags de S-PEP. Fuente: propia.

6.1.3.3.1 SYN

Si se habilita este flag, se indica al receptor del segmento que el cliente desea iniciar una conexión.

6.1.3.3.2 ACK

Si este flag está habilitado, el emisor del segmento está indicando al receptor que ha recibido un segmento.

6.1.3.3.3 FIN

Si este flag está habilitado, el emisor del segmento está indicando que desea finalizar la conexión.

6.1.3.3.4 RST

Si este flag está habilitado, el emisor desea reiniciar la conexión de forma unilateral. No debe usarse a menos que haya un error severo en la conexión.

6.1.3.3.5 ARR

Si este flag está habilitado, el emisor está indicando al receptor que hay una reducción pronunciada del Allocation Size.

6.1.3.3.6 MCD

Si este flag está habilitado, uno de los dispositivos de enrutamiento con los que se ha cruzado el segmento ha indicado que hay congestión en la red.

6.1.3.3.7 NACK

Si este flag está habilitado, el emisor está haciendo explícito que no ha recibido confirmación de un segmento.

6.1.3.4 Sequence Number (32 bits)

El número de secuencia es un campo de 32 bits que indica el número de segmento que el emisor está enviando. Es inicializado en el proceso de establecer una conexión, y con cada segmento enviado, el número aumenta.

6.1.3.5 Acknowledge Number (32 bits)

El número de reconocimiento de recepción (ACK) es un campo de 32 bits que indica al receptor el número de secuencia que está esperado recibir.

6.1.3.6 Allocation Window Number (32 bits)

Este campo indica al receptor la cantidad de segmentos que el emisor está dispuesto a recibir. El receptor de este segmento puede enviar tantos segmentos quiera hasta que su número de secuencia sea igual al Allocation Window Number.

Se ha optado por utilizar este método y no indicar la cantidad de bytes que puede aceptar, tal y como hace TCP, porque es un método más simple para emisor y receptor (además de posiblemente más rápido) de calcular el máximo de paquetes que pueden recibir sin ACKs.

6.1.3.7 Connection Process

El proceso de conexión es similar al de TCP. El cliente debe mandar un segmento con el flag SYN habilitado, el servidor debe contestar con los flags SYN y ACK, y el cliente debe comunicar la recepción del mensaje mediante el flag ACK.

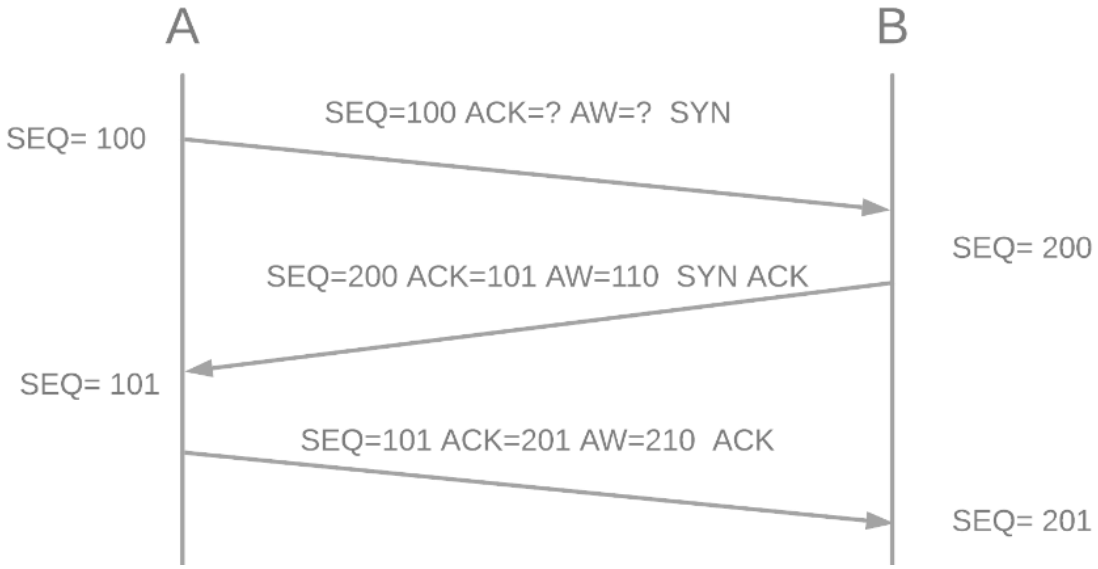


Figura 28 Diagrama de una conexión S-PEP. Fuente: propia.

6.1.3.8 Disconnection Process

El proceso de desconexión es similar al de TCP. El cliente solicita una desconexión mediante el uso del flag FIN. El servidor debe contestar al cliente con el flag FIN y el flag ACK habilitados, indicando que se dispone a cerrar la conexión, y el cliente debe notificar mediante el flag ACK la recepción de la desconexión.

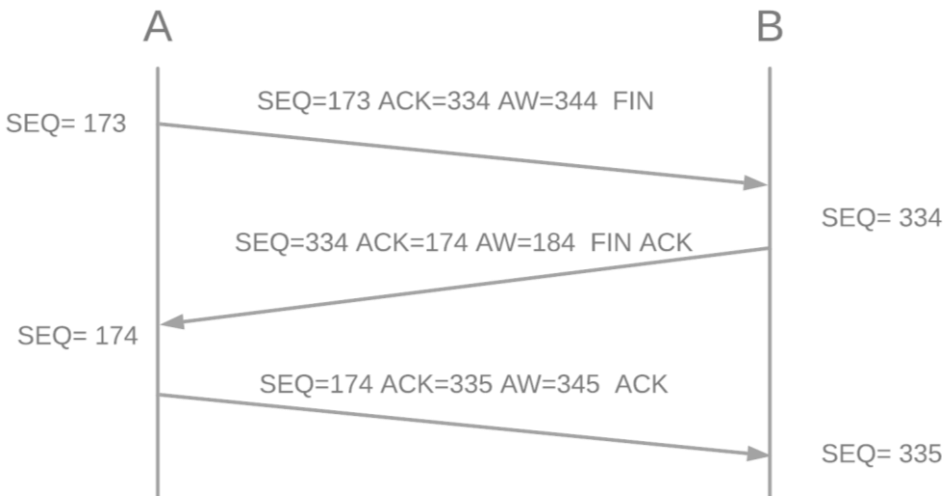


Figura 29 Diagrama de una desconexión S-PEP. Fuente: propia.

6.1.3.9 Sliding Window

Gracias al campo Allocation Size, y al igual que en TCP, S-PEP permite usar Sliding Window para enviar múltiples segmentos sin esperar la confirmación explícita de su recepción. S-PEP permite usar tanto los algoritmos Go-Back-N como Selective Repeat. Por defecto, S-PEP utiliza Go-Back-N para conseguir el efecto de una Sliding Window.

Cuando se utiliza Go-Back-N la pila de red permite enviar tantos paquetes como se requieran hasta que el número de secuencia local sea igual al último Allocation Window recibido. Esto permite que las confirmaciones de recepción vayan llegando en paralelo, aumentando así la eficiencia de S-PEP.

En caso de que un paquete se pierda, como sucede en el diagrama, el receptor descarta todos los paquetes recibidos a continuación porque el número de secuencia esperado es menor al número de secuencia recibido.

Cuando el emisor lleva mucho tiempo sin recibir una confirmación de recepción, o si la Allocation Window Number se llena, el emisor vuelve a enviar los segmentos desde la última confirmación de recepción recibida.

En caso de que lo que se pierda sea una confirmación de recepción y, por ende, el número de ACK recibido sea mayor al esperado, el emisor vuelve a enviar los segmentos desde el último ACK correcto recibido. El receptor, entonces, deberá volver a enviar las confirmaciones de recepción hasta el número de secuencia esperado.

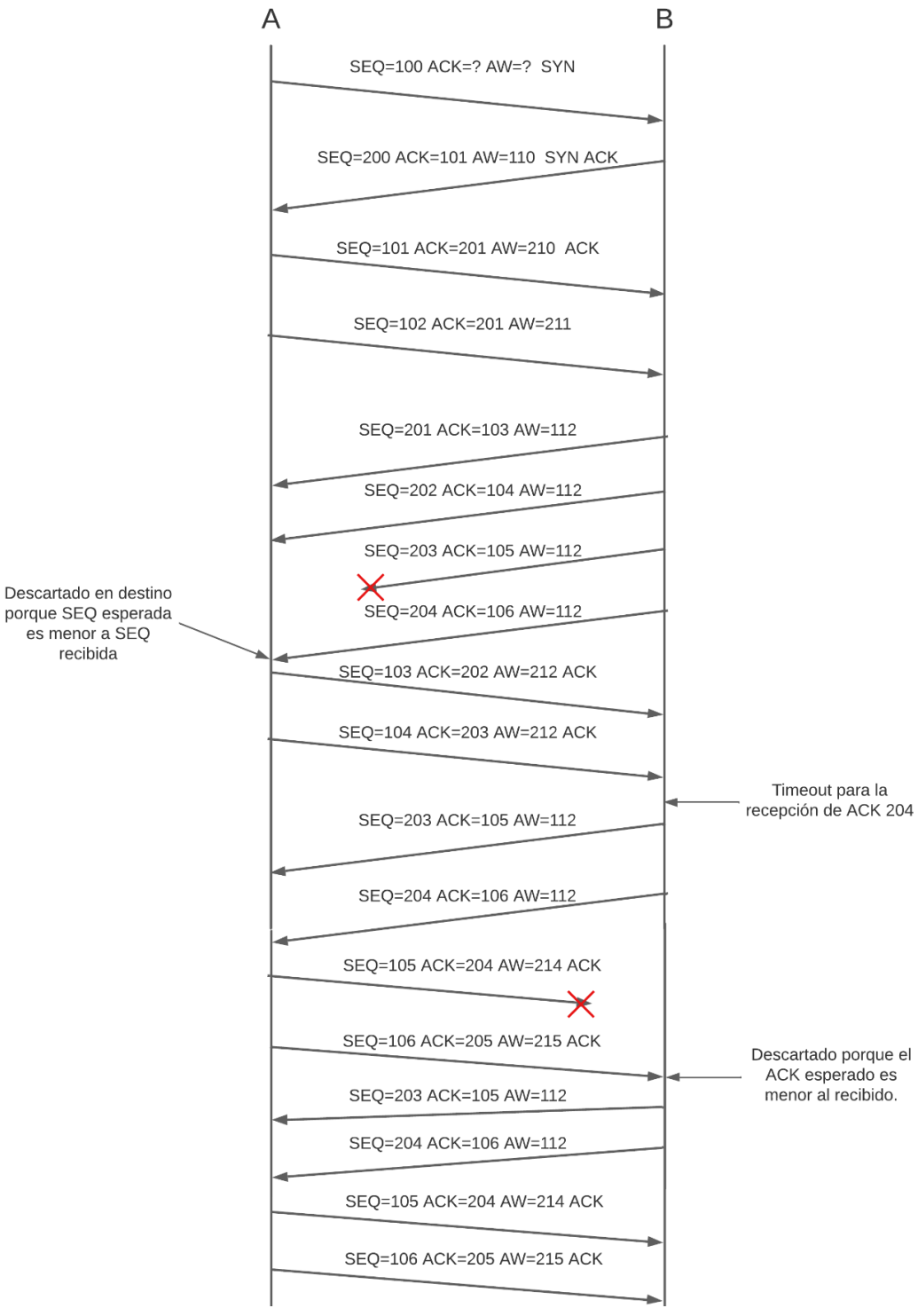


Figura 30 Diagrama de Sliding Window en S-PEP. Fuente: propia.

6.1.4 Dynamic Network Discovery Protocol (DNDP)

El protocolo DNDP es parte esencial de la pila de red IPDP; ya que es el encargado de descubrir redes IPDP, crearlas, y anunciarlas.

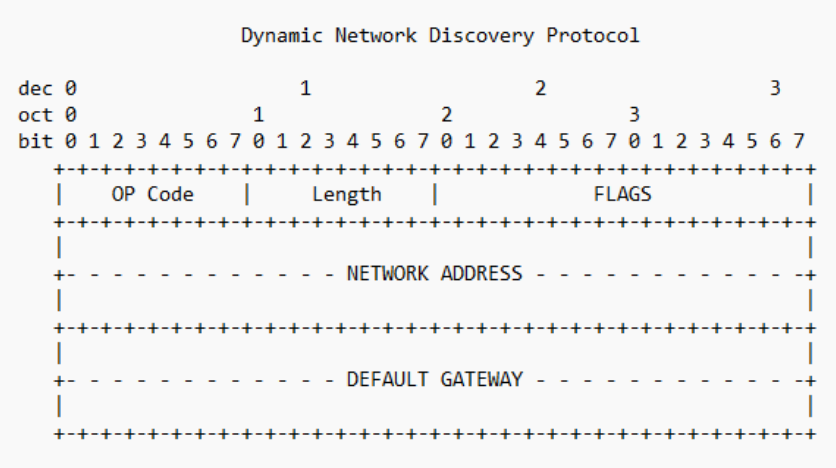


Figura 31 Cabecera de un paquete IPDP. Fuente: propia.

Este protocolo consta de una cabecera de dieciséis bytes con cinco campos que son transmitidos en broadcast a toda la red.

6.1.4.1 OP Code (8 bits)

El primer campo consiste en un byte dedicado al código de operación. En la primera especificación de este protocolo solo existen tres códigos posibles, descritos a continuación.

OP CODE	Meaning
0x01	Discover IPDP Network.
0x02	Advertise IPDP Network.
0xFF	DNDP Error. Please, try again.

Figura 32 OP Codes DNDP. Fuente: propia.

6.1.4.2 Length (8 bits)

Este campo indica el tamaño de la dirección en bytes. Este campo puede ser aparentemente inútil, ya que todas las direcciones IPDP contienen solo 64 bits de dirección de red en su modo normal de operación; pero por cuestiones de compatibilidad futura (además de aumentar la compatibilidad con otros protocolos) se ha decidido incluirlo.

6.1.4.3 Flags (16 bits)

El primer bit de este campo indica si se dispone de una puerta de enlace por defecto o no, en caso de que se disponga se adjuntará después de la dirección de red devuelta.

El segundo bit indica si se requiere usar direcciones IPDP extendidas, en caso de que así sea, las direcciones devueltas son de 128 bits.

6.1.4.4 Direcciones de red y puerta de enlace

Estos dos campos indican las direcciones de red (de 64 o 128 bits) y la dirección física de la puerta de enlace por defecto en caso de que el flag correspondiente esté activado.

6.1.4.5 Proceso de Discover

Hay tres casos posibles para un proceso de Discover. El primero es que el dispositivo sea el primero en tratar de hacer un Discover sin que haya ningún otro equipo conectado.

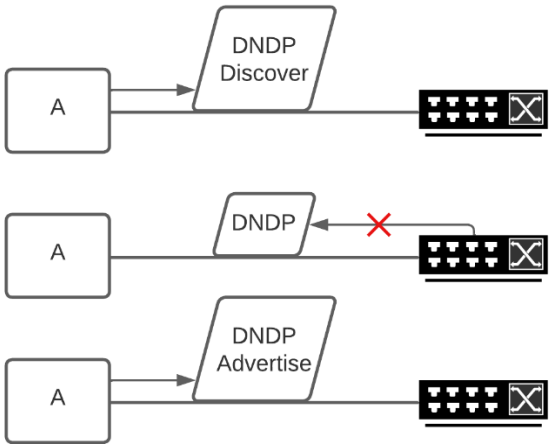


Figura 33 Diagrama Discover. Fuente: propia.

En este caso, al no recibir respuesta alguna a esta petición, el host creará una nueva red IPDP (generando la dirección de red de forma aleatoria) y la anunciará. En caso de que un nuevo Advertise le llegue al host, este decidirá si desestimarlo o volver a realizar el proceso de Discover.

El segundo caso consiste en que el dispositivo se conecte a una red donde previamente ya había una red IPDP establecida.

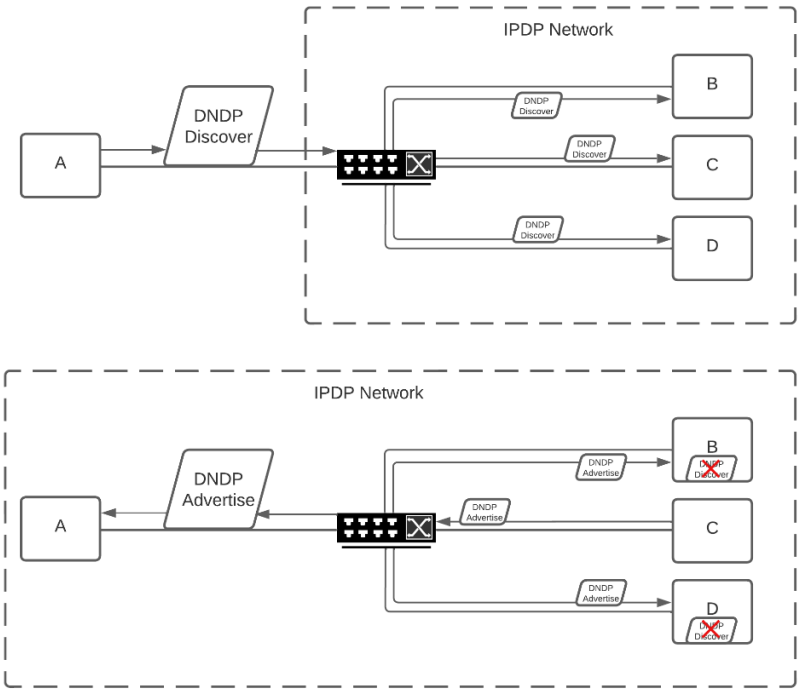


Figura 34 Diagrama Discover. Fuente: propia.

En este caso, todos los dispositivos que reciben la solicitud deben iniciar una cuenta atrás de un número aleatorio de segundos (entre cero y diez). Si cuando la cuenta atrás se ha terminado, no se ha recibido ningún paquete de Advertise, el dispositivo debe enviar uno con la información de la red IPDP.

Por último, existe la posibilidad de que el ISP quiera proporcionar una dirección de red a la red IPDP.

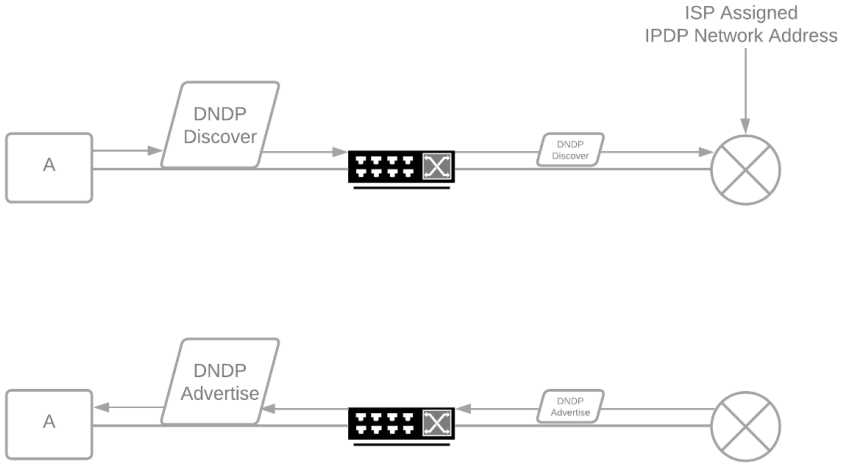


Figura 35 Diagrama Discover. Fuente: propia.

En este caso, el router que tiene la dirección IPDP debe responder de inmediato a la solicitud de Discover.

6.1.5 Routing Information Exchange Protocol (RIEP)

RIEP es un protocolo muy similar en funcionamiento a RIP. La principal diferencia reside en la estructura de los paquetes y en que es capaz de aceptar diferentes tipos de direcciones y protocolos.

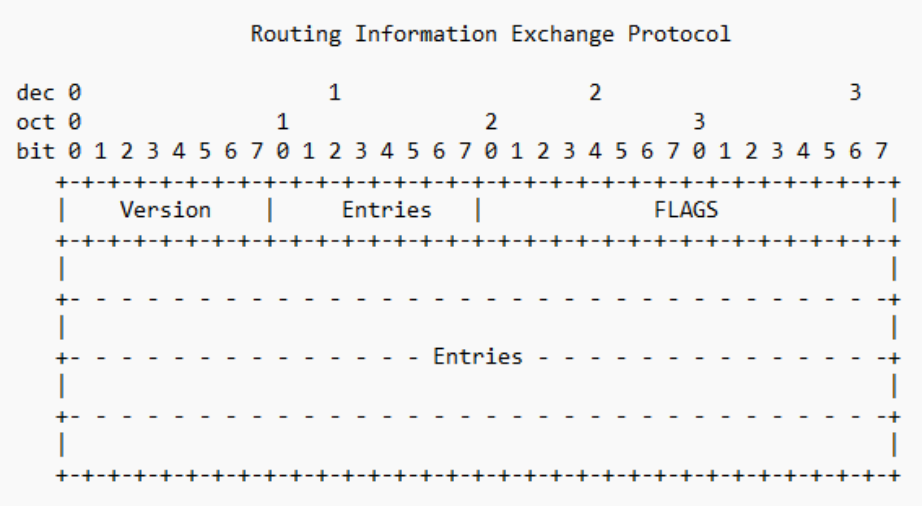


Figura 36 Estructura de un paquete RIEP. Fuente: propia.

Este protocolo se utiliza entre dispositivos enrutadores para propagar la información de las rutas disponibles y la distancia hasta ellas. De esta forma, los routers pueden tener la información de enrutamiento de forma dinámica,

Cada router tiene una pequeña base de datos con los registros RIEP que tiene disponibles.

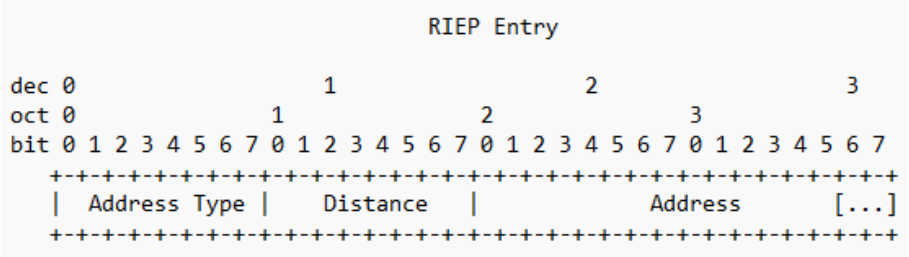


Figura 37 Diagrama de un registro RIEP. Fuente: propia.

Estos registros constan del tipo de dirección (ej. IP, IPX, IPDP), la distancia hasta esa dirección, y la dirección en si (seguida o no de la máscara de red en función del tipo de dirección).

Este protocolo hace que los routers envíen la lista de registros RIEP de las conexiones que tiene directamente conectadas en forma de broadcast cada minuto.

Mediante el uso de los FLAGS, es posible solicitar a un router cercano su base de datos RIEP sin necesidad de esperar al broadcast del resto de routers.

6.1.6 Service Advertisement Protocol (SAP)

Service Advertisement Protocol es un elemento principal de la suite de protocolos IPDP. Este protocolo se encarga de informar a la red local de los servicios de red disponibles a todos los equipos.

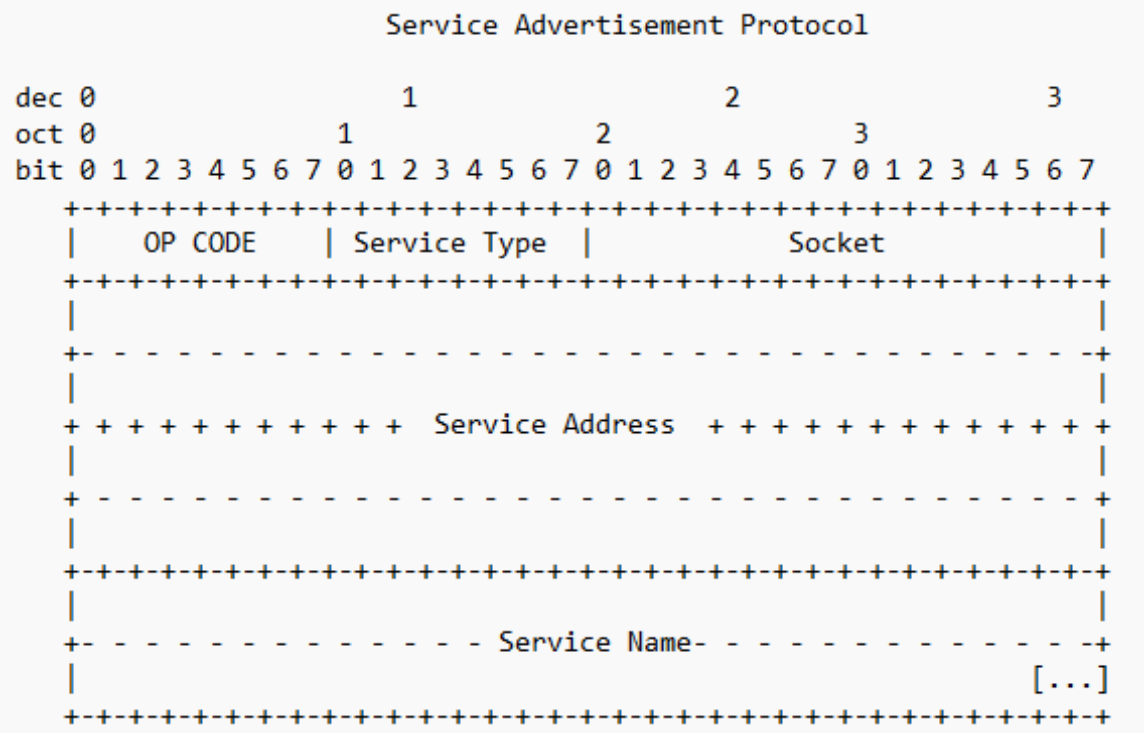


Figura 38 Estructura de un paquete SAP. Fuente: propia.

Su cabecera consta de cinco campos, que en total suman los 28 bytes que ocupa un paquete SAP convencional.

6.1.6.1 OP Code (8 bits)

Este campo de un byte indica el código de operación del paquete SAP. Este código puede indicar que se trata de un nuevo servicio (0x01), de una edición a un servicio ya disponible (0x02), de una baja de un servicio (0x04), o de una solicitud de recepción de los servicios (0xFF).

6.1.6.2 Service Type (8 bits)

Este campo de 8 bits indica el tipo de servicio que se presta. En caso de que el bit más significante de este campo esté habilitado (que contenga un uno), la dirección IPDP pasa de 128 bits a 192 bits, ya que se está usando el modo IPDP extendido.

6.1.6.3 Socket (16 bits)

Este campo de 16 bits indica el socket donde se presta el servicio.

6.1.6.4 Service Address (128 o 192 bits)

Este campo de 128 o 192 bits contiene la dirección IPDP del servicio prestado.

6.1.6.5 Service Name

Este campo de tamaño variable indica el nombre del servicio. El tamaño de este campo comprende desde el primer bit después del último bit de la dirección hasta el fin del paquete; por lo que la longitud del campo viene dada por la MTU de la red.

6.1.7 Route Reservation Protocol (RRP)

Este protocolo es un protocolo orientado a la prestación de servicios que requieran de no solo baja latencia, sino de una variación de esta (jitter) constante. Su principal función es la de solicitar a los routers entre dos hosts que mantengan, para ese cliente, siempre la misma ruta; además de solicitar preferencia en la transacción de los paquetes de la ruta.

Cuando se realiza una solicitud RRP, los routers deben enviar el paquete hasta que llegue al destino. En caso de que en algún momento un router no pueda aceptar esta reserva, deberá interrumpir la propagación de la solicitud y enviar al emisor un mensaje MCP indicando el error.

Cada router por el que pasa la solicitud RRP debe revisar el contenido del campo MTU del paquete; ya que si una de las redes por las que debe transitar el paquete tiene un MTU menor al especificado en ese campo, el router tiene la obligación de sobrescribirlo e indicar el nuevo tamaño máximo de la MTU.

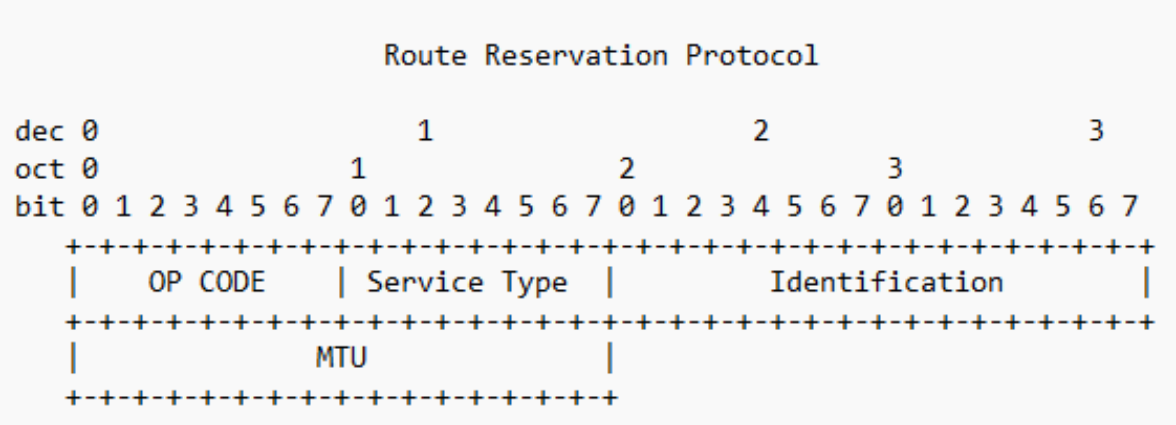


Figura 39 Diagrama de un paquete RRP. Fuente: propia.

6.1.7.1 OP Code

Este campo indica el código de operación RRP. Puede ser desde una solicitud RRP (0x04) hasta la cancelación del servicio RRP (0x06).

6.1.7.2 Service Type

Este campo del paquete RRP indica el tipo de servicio para el que se va a usar esta reserva. Puede ser desde contenido multimedia en vivo, hasta videoconferencias o juegos en línea. Esto ayuda a los dispositivos de enrutamiento a adaptarse al tipo de tráfico que va a circular por la ruta reservada.

6.1.7.3 Identification

Este campo indica a todos los routers (y a la máquina destino) cuál va a ser el código de identificación que van a usar los paquetes de ese flujo reservado.

6.1.7.4 MTU

Este campo indica el tamaño máximo de los paquetes que pueden circular por ese flujo de paquetes.

6.1.8 Message Control Protocol (MCP)

El protocolo MCP es parte fundamental de la pila de red IPDP, ya que de él dependen otros protocolos para funcionar. Su principal función es la de informar de diversos eventos a los dispositivos de una red. Estos eventos pueden ser desde errores, hasta denegaciones a una petición.

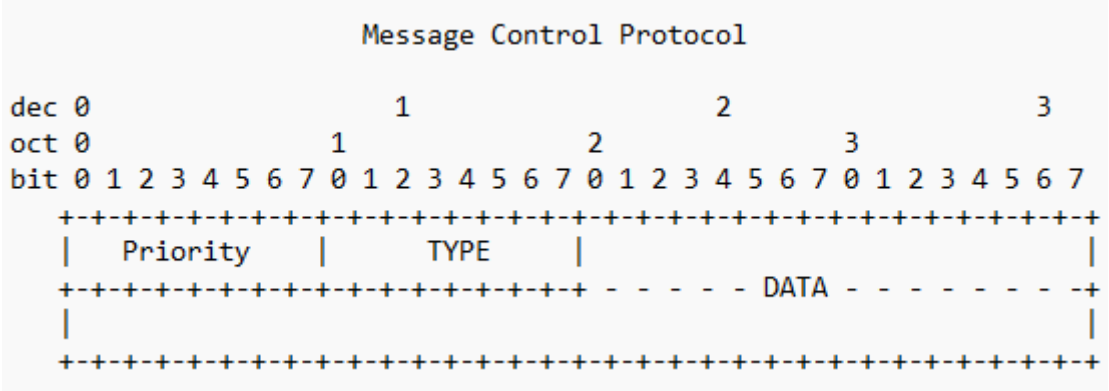


Figura 40 Diagrama de un paquete MCP. Fuente: propia.

Un paquete MCP consta de tres campos esenciales.

6.1.8.1 Priority (8 bits)

Este campo de un byte indica la severidad o la prioridad del mensaje a transmitir. La prioridad va desde la más baja, cero, hasta la más alta, 255. Esta prioridad se determina mediante el tipo de mensaje a transmitir, su importancia, y las veces que se ha repetido esa condición.

6.1.8.2 Type (8 bits)

Este campo de 8 bits indica el tipo de mensaje que se está enviando. El mensaje puede ser desde una solicitud de un eco, hasta la información expresa de que hay congestión en una red, pasando por cosas como que un destino es inalcanzable o que la red destino no es alcanzable desde la red actual.

6.1.8.3 Data

Este campo contiene datos que pueden ser relevantes para el receptor. Puede ayudar a localizar un problema, indicar un número de dispositivos, o dar una explicación más detallada de un error.

6.2 Diseño del software

En este apartado se discuten las cuestiones tecnológicas básicas para poder trasladar el diseño anterior a un software robusto.

6.2.1 Escogiendo un paradigma

Antes de poder empezar a programar la pila de red hay que hacerse una pregunta fundamental: ¿qué conjunto de herramientas debo usar para programación de sistemas?

Una respuesta perfectamente válida a esta pregunta sería mencionar C y C++ como los lenguajes de facto para este tipo de tarea. Otras personas mencionarían Rust y Carbon, quizá incluso Go; sin embargo, pocas personas mencionarían a un lenguaje que requiere de una VM para funcionar, como C#. Sin embargo, las características de seguridad que puede aportar un lenguaje como éste (en cuanto a seguridad de tipos, de memoria, etc.) pueden ser más beneficiosas que las aparentes desventajas.

A esta conclusión llegaron los programadores que estuvieron dedicando su tiempo entre 2007 y 2014 (Duffy, Blogging about Midori, 2015) a Project Midori.

6.2.1.1 Project Midori

Project Midori fue un sistema operativo, que nunca vio la luz de forma comercial, desarrollado por más de 100 desarrolladores en Microsoft. Fue desarrollado por Microsoft Research, y su misión era la de crear un sistema operativo escrito íntegramente en managed code (Foley, 2015).

Durante el desarrollo de Midori se comprendió que no se podía usar solo C#, en el estado en el que estaba en aquella época, para lograr hacer un buen sistema operativo, así que el equipo que trabajaba en el proyecto creó su propio lenguaje de programación (basado en C#), su propio compilador, y sus propias herramientas para lograr su objetivo.

Cuando el proyecto se finalizó, todos estos cambios y aprendizajes que hicieron con esa versión ampliada de C# fueron portados a .Net y a las “mejores prácticas” de C++ (Duffy, Joe Duffy's Blog, 2015).

6.2.1.2 Ixy.cs

Unos años después, (Stadlmeier, 2018) trató de mostrar la viabilidad de .Net Core para la programación de sistemas al probar a portar el controlador IXY para una tarjeta de red Intel 10GbE en C#. En su tesis explica como el rendimiento de esta versión del controlador es, en la mayoría de los casos, comparable a la de C.

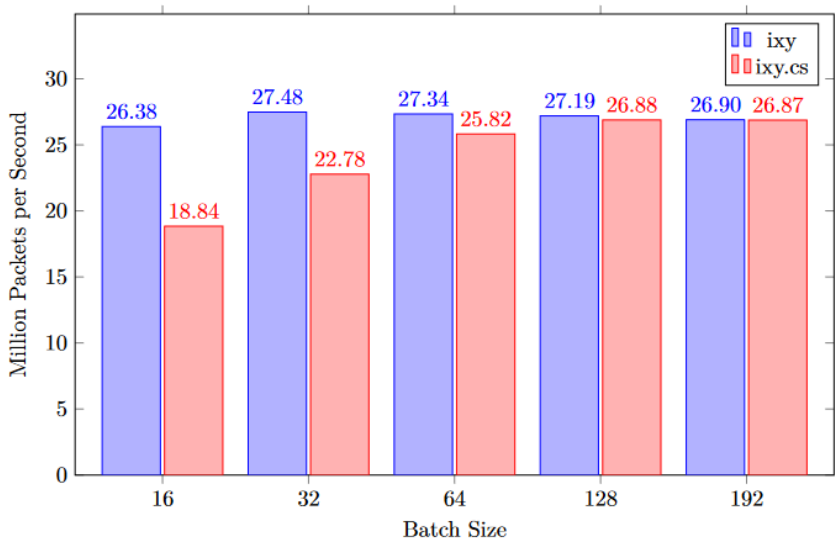


Figura 41 Gráfica comparativa entre el rendimiento de ixy e ixy.cs. Fuente: (Stadlmeier, 2018).

6.2.1.3 C# como lenguaje de programación de sistemas

Con las recientes adiciones del equipo de .Net al lenguaje, y las constantes mejoras del runtime, se espera que la pequeña brecha entre el rendimiento de C y C# pueda cerrarse aún más.

Además, se espera que las ventajas de tener seguridad de tipos y de memoria permita la creación de un código robusto y de más fácil mantenimiento.

Por estas razones, y con estos dos ejemplos como guía, se ha decidido utilizar C# como lenguaje vehicular para este proyecto.

Capítulo VII: Análisis de resultados

El desarrollo del proyecto está dividido en dos grandes bloques: el primero consiste en el desarrollo de un Userspace Driver capaz de recibir y enviar tramas ethernet, el segundo consiste en la creación de un SDK para poder utilizar IPDP en cualquier aplicación que soporte .Net.

El esquema básico de la pila de red de un juego de Unity consta de cuatro elementos clave: un Miniport Driver, que se encarga de comunicarse con el hardware del NIC; un Protocol Driver, que se encarga de aportar las peculiaridades del stack TCP/IP a las tramas obtenidas del Miniport Driver, a la vez que provee de paquetes IP al Miniport Driver para que sean enviadas; el “SDK” WinSocks 2, una capa de abstracción sobre el Protocol Driver con el que las aplicaciones pueden comunicarse con la red; y la propia implementación que se decida hacer de WinSocks 2 en el motor Unity.

Figura 42 Diagrama del esquema simplificado NDIS. Fuente: propia.

Para poder crear una pila de red alternativa, se necesita un punto de entrada en el esquema mencionado anteriormente; en este caso, el punto escogido como punto de entrada reside entre el Miniport Driver y el Protocol Driver. Esto es debido a que es el único punto de la pila en la que se tiene un acceso de un cierto alto nivel al NIC, ofrecido por el Miniport Driver, pero aún no se ha dado por supuesto que todo el tráfico que va a circular por la pila es puramente IP.

Para poder tener un punto de entrada en este punto, se necesita de un Filter Driver que exponga dos funciones clave: Enviar y Recibir. El Filter driver escogido es NPCAP, un Filter driver muy conocido en el mundo del análisis de tráfico de red, ya que es un elemento principal del popular programa Wireshark.

Una vez localizado un punto de entrada adecuado, se ha desarrollado toda la pila alternativa alrededor de este.

Los elementos principales de esta pila alternativa son: el IPDP Userspace Driver, que se encarga de enviar y recibir los paquetes IPDP, además de incorporar un pequeño cliente DNDP para descubrir redes IPDP y, en caso de que no exista una, crearla; y una API al estilo WinSocks 2 que permita comunicarse entre el proceso del controlador y los diferentes programas que quieran enviar y recibir tráfico IPDP.

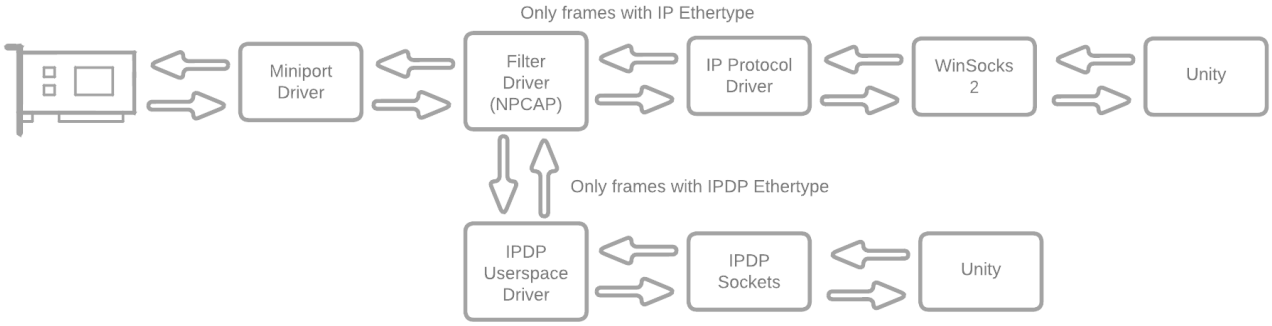


Figura 43 Diagrama de la solución propuesta. Fuente: propia.

7.1 IPDP Userspace Driver

La filosofía escogida para programar el controlador en espacio de usuario; es decir, un controlador que no requiere de permisos de administrador ni de acceso directo al kernel de Windows; ha sido la de minimizar las recolecciones de basura al máximo.

Para conseguir aproximarse lo máximo posible a la meta de cero kilobytes de memoria dinámica recolectados; haciendo así que exista una presión nula en el recolector de basuras, y que, por ende, este nunca llegue a hacer recolecciones continuas durante la vida del proceso; se ha optado por el uso de estructuras como el principal contenedor de datos del proyecto; además de dejar ligeramente de lado la visión purista de la programación orientada a objetos.

7.1.1 La importancia del recolector de basura

En un programa escrito en cualquiera de los lenguajes soportados por .Net existe un elemento en el CLR (Common Language Runtime) llamado Recolector de Basura (GC o Garbage Collector) (Microsoft, 2023). Este elemento se encarga de la asignación y la liberación de memoria de forma automática y transparente para el programador, además de efectuar tareas de compactación, reordenación, y optimización de la memoria.

Este mecanismo no solo busca acabar con varios de los problemas clásicos de la programación; como es el uso de un puntero después de ser eliminado (use after free), la doble eliminación de un puntero, o la no eliminación de segmentos de memoria que no son apuntados por ningún puntero (memory leak).

Para que se produzca una recolección de basura automática; es decir, una recolección de basura no producida por la instrucción GC.Collect(); se requiere de unas condiciones específicas que dependen, o no, de la propia aplicación.

Si el sistema operativo notifica a la aplicación de .Net que está quedando sin memoria RAM disponible, la aplicación lanzará una recolección de basura para tratar de aliviar la presión sobre la memoria del sistema.

A su vez, si la aplicación detecta que la cantidad de memoria asignada en el Heap supera un lindar específico, el runtime de .Net lanzará una recolección de basura para tratar de aliviar la presión sobre la memoria del sistema, aumentando así la presión sobre el recolector de basura.

A simple vista, aumentar la presión sobre el recolector de basura no parece una mala idea en sistemas donde minimizar el uso de la memoria es algo primordial. Sin embargo, el aumento de presión en este mecanismo tiene consecuencias en el rendimiento y la latencia de la aplicación. Para comprender mejor las consecuencias del uso excesivo de este mecanismo hace falta entender cómo funciona concretamente una recolección de basura.

Todo elemento de la memoria Heap, o la memoria destinada para su uso dinámico, en ,Net está en una de las siguientes tres generaciones del recolector de basura:

- Generación cero: Es esta generación se encuentran los objetos acabados de crear. Cuando un objeto es creado (si su tamaño no es extremadamente grande, que entonces pasa directamente a la generación dos), o un segmento de memoria es asignado, el recolector de basura (que también es el encargado de la creación de objetos y la asignación de memoria) marca esa región de forma automática como una región de generación cero.

- Generación uno: En esta generación se encuentran los objetos que han sobrevivido a una recolección de basura cuando estaban en la generación cero. El mecanismo de recolección de basura de .Net dicta que cualquier objeto que sobreviva a una recolección de basura, si no se encuentra ya en la generación dos, debe ser promovido a la siguiente generación.

- Generación dos (y LOH): En esta generación se encuentran tanto los objetos más antiguos como los más grandes.

Cuando se realiza una recolección de basura, el GC trata de reducir el uso de memoria de forma secuencial entre las diferentes generaciones. De este modo, si una recolección de basura de generación cero es suficiente para aliviar la presión de memoria, el recolector de basura finalizará allí su intervención.

La recolección de memoria de cada generación se divide en tres fases principales; en la primera fase se escanea, normalmente en paralelo con la ejecución de la aplicación, la memoria Heap con la intención de buscar aquellos objetos que no son apuntados por ninguna variable; en la segunda fase se buscan oportunidades de compactar la memoria (si es posible); y en la tercera fase es cuando el recolector de basura debe pausar todos los hilos de la aplicación y realizar los cambios necesarios en la memoria para liberar espacio y compactarla; moviendo las referencias en caso de que se haya tenido que mover objetos.

La importancia de no aumentar la presión en el recolector de basura reside en la fase tres de una recolección, donde se pausan los hilos de la aplicación. En un programa normal, el tiempo que los hilos de una aplicación se encuentran pausados es (si la aplicación está bien programada) prácticamente imperceptible, pero en el caso de un software que tiene que ejecutarse lo más rápido posible, para minimizar la latencia y evitar perder paquetes, es esencial mantener lo más baja posible la presión sobre el GC.

7.1.2 Recepción de tramas

Para la recepción de tramas ethernet con el EtherType adecuado se ha utilizado SharpPcap, una librería para .Net que actúa como wrapper para la librería, escrita en C++, Npcap. Esta librería es la encargada de proporcionar acceso al Filter driver, que es el punto de entrada en la cadena NDIS escogido para enviar y recibir tramas ethernet.

Los dispositivos elegibles para poder usar la pila de red IPDP son, en este prototipo, las tarjetas de red Ethernet conectadas a algún dispositivo. Para poder localizar los adaptadores que cumplen estas condiciones, es necesario realizar la siguiente operación.

```
ethNics = LibPcapLiveDeviceList.Instance.Where(x:LibPcapLiveDevice =>
    !((((InterfaceFlags)x.Interface.Flags).HasFlag(InterfaceFlags.PcapIfWireless)) ||
      ((InterfaceFlags)x.Interface.Flags).HasFlag(InterfaceFlags.PcapIfLoopback) ||
      ((InterfaceFlags)x.Interface.Flags).HasFlag(InterfaceFlags.PcapIfConnectionStatusDisconnected)) &&
    ((InterfaceFlags)x.Interface.Flags).HasFlag(InterfaceFlags.PcapIfUp) &&
    ((InterfaceFlags)x.Interface.Flags).HasFlag(InterfaceFlags.PcapIfRunning) &&
    !x.Description.Contains("iport")).Select(x:LibPcapLiveDevice => (IpdpNetworkAdapter)x).ToArray(); // IpdpNetworkAdapter[]
```

Figura 44 Filtro de adaptadores de red. Fuente: propia.

Una vez se han encapsulado los adaptadores de red en IpdpNetworkAdapters, se puede proceder a la recepción de las tramas ethernet.

Para recibir las tramas ethernet se usa la siguiente función de SharpPcap.

```
public int GetNextPacketPointers(ref IntPtr header, ref IntPtr data)
{
    return LibPcapSafeNativeMethods.pcap_next_ex(Handle, ref header, ref data);
}
```

Figura 45 Código de GetNextPacket de SharpPcap. Fuente: propia.

Se utiliza este método porque las otras versiones utilizadas por la librería crean copias de los paquetes en la memoria Heap para evitar perder información; ya que este método solo garantiza la validez de los datos apuntados por el puntero data hasta la siguiente llamada. En caso de que no haya datos disponibles para leer el método retornará cero, y en caso de que sí que haya datos por leer el método retornará el número de bytes disponibles.

Otro de los principios que se ha usado a la hora de escribir el código de este programa es el de minimizar las copias entre buffers. A pesar de que el objetivo era quedar lo más cerca posible del “Zero Copy IO” de Linux, desde un principio se tomó por imposible semejante objetivo y se procedió a establecer el límite en dos copias íntegras de la trama o del paquete.

Debido a este principio, todo el proceso entre que se recibe una trama y esta es enviada a un programa debe ser lo más rápido y breve posible.

7.1.3 Network Order y Host Order

El endianismo es la forma en la que un procesador interpreta los valores de más de un byte que deben entenderse como un conjunto (Wong, 2018). Un ejemplo de esto es el valor almacenado en una variable de tipo UInt32 (entera de 32 bits sin signo). En un sistema Little endian, el byte menos significante estará primero en la secuencia de bytes; es decir, si se quisiera interpretar el valor 255 (en decimal) almacenado en una variable UInt32, el resultado de la cadena de bytes sería 0xFF000000.

En el mundo de las redes, el formato escogido para transmitir los datos fue Big endian (Arora, 2011); es decir, si se tratase de transmitir el mismo entero sin signo del ejemplo anterior se transmitiría de la siguiente forma: 0x000000FF. Esto resulta en un problema a la hora de tratar de recibir e interpretar los datos de las tramas de red recibidas.

La forma más simple de cambiar el endianismo a una estructura de datos, es crear una estructura de datos con el orden de los campos invertido, asignar los valores a sus nuevas posiciones, e invertir la secuencia de bytes resultante. Si se toma como ejemplo los primeros cuatro bytes de la cabecera IPDP, el proceso a realizar es el siguiente.

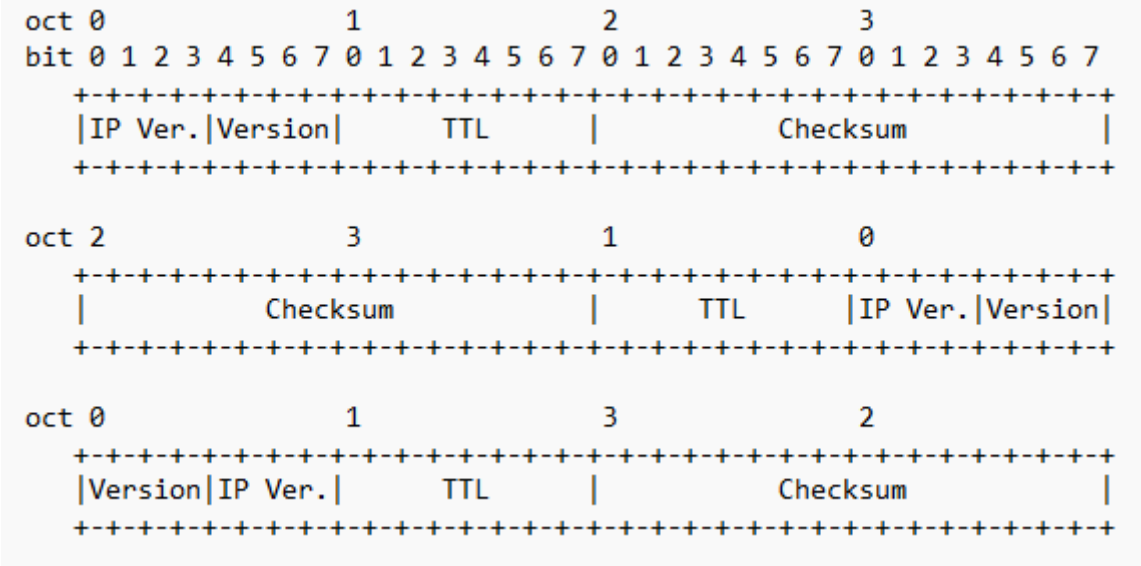


Figura 46 Proceso de cambio de Endianismo. Fuente: propia.

Como se puede observar en la figura anterior, se parte con una estructura de datos que contiene cuatro campos en cuatro bytes. Estos campos son la IP Version, que conforma los cuatro primeros bits del primer byte; la Versión de IPDP, que conforma los siguientes cuatro bits; el TTL, que corresponde al siguiente byte; y el Checksum que ocupa los dos últimos bytes.

Estos campos son asignados a sus nuevas posiciones en la estructura invertida, quedando el Checksum en la dos primeras posiciones, el TTL en la tercera, y el byte con las versiones en último lugar.

Una vez asignados los campos, se interpreta la estructura entera como una secuencia de bytes y se invierte (a nivel de bytes). De esta forma, queda en primera posición el byte de las versiones, en segunda posición el TTL, en tercera posición el último byte del Checksum, y en cuarta posición el primer byte del Checksum. De esta forma, si el Checksum fuera 0xFAFE en little endian, en big endian sería 0xFEFA.

La decisión de mantener el endianismo Big endian hasta la capa tres se ha tomado, principalmente, por cuestiones de retrocompatibilidad. En caso de que IPDP sea ejecutado en una red que comparte tráfico con IP, y la capa dos sea de tipo 802.3 (o derivados) en vez de Ethernet II, los primeros campos de la cabecera IPDP son compatibles con los de la cabecera IP, por lo que los dispositivos entenderían que se trata de una versión desconocida de IP e ignorarían los paquetes sin causar conflicto alguno.

En caso de encontrarse con una red Ethernet II (que hoy en día son las más comunes), el campo EtherType ya se encarga de diferenciar el tráfico encapsulado.

A partir de la capa tres de la suite desarrollada, las cabeceras y los datos encapsulados son interpretados con el endianismo Little Endian, haciendo más fácil y rápido el envío y la recepción de los datos.

7.1.3 Envío de tramas

Para el envío de tramas ethernet se ha utilizado el método SendPacket de SharpPcap. Este método requiere un array o un span, una estructura constituida por una referencia y un tamaño máximo.

```
IL code
public void SendPacket(ReadOnlySpan<byte> p, ICaptureHeader header = null)
```

Figura 47 Cabecera del método SendPacket. Fuente: propia.

El método SendPacket no dista de ser un wrapper alrededor de una llamada al Unmanaged DLL NPCAP, como se puede observar en la siguiente figura.

```
[DllImport(PCAP_DLL, CallingConvention = CallingConvention.Cdecl)]
IL code
internal extern static int pcap_sendpacket(PcapHandle /* pcap_t* */ adaptHandle, IntPtr data, int size);
```

Figura 48 DLLImport de SendPacket. Fuente: propia.

Este método espera que el span o array enviado contenga los elementos de una cabecera ethernet, por lo que no provee forma de crearlos antes de enviarlos. Por este motivo, se requieren de catorce bytes adicionales al tamaño del paquete a enviar, que nunca puede ser superior al MTU.

Los catorce bits que preceden a la cabecera IPDP conforman la cabecera de una trama Ethernet II. En caso de que se requiera el uso de otra tecnología, el tamaño reservado para esta cabecera puede variar.

Los primeros doce bytes corresponden a la MAC destino y la MAC origen. Tras esto, está el EtherType de IPDP (0x88B4) en formato Big Endian.

Para el envío de tramas se ha querido utilizar una filosofía similar al de la recepción de estas, tratando de realizar el menor número de copias posible; sin embargo, ha sido imposible cumplir del máximo de dos copias enteras, teniendo que aumentar el número a tres copias íntegras debido a problemas entre el recolector de basura y la gestión de memoria realizada por NPCAP.

Para poder enrutar correctamente los paquetes a la salida se ha construido la clase IpdpNetworkAdapter, mencionada anteriormente, que encapsula un adaptador de red,

su dirección IPDP y la ruta por defecto, si es que existe. Mediante un diccionario de direcciones de red IPDP (la primera mitad de una dirección IPDP entera) y su respectivo IpdpNetworkAdapter, se puede dirigir los paquetes hacia la respectiva ruta por defecto.

En apartados posteriores se detallarán las características del enrutado de paquetes.

7.1.4 Calculando el checksum

En la especificación de IPDP se especifica que, si el flag CHK está habilitado, todo el paquete se debe someter a una operación de CRC16 para obtener el checksum.

Para calcular el CRC16 del paquete se exploró el uso de diversos paquetes, entre ellos System.Data.HashFunction.CRC y NullFX.CRC. Fue una grata sorpresa descubrir que el propio desarrollador del paquete NullFX ya había preparado un benchmark con BenchmarkDotNet mostrando las diferencias entre ambos paquetes.

Method	ArraySize	Mean	Error	StdDev	Gen 0	Allocated
NullFxCrc16	10	14.21 ns	0.300 ns	0.280 ns	-	-
DataHashFunctionCrc16	10	256.61 ns	5.061 ns	5.197 ns	0.0362	304 B
NullFxCrc16	100	203.19 ns	3.947 ns	3.692 ns	-	-
DataHashFunctionCrc16	100	861.40 ns	17.206 ns	28.269 ns	0.0362	304 B
NullFxCrc16	1000	2,125.76 ns	26.518 ns	23.507 ns	-	-
DataHashFunctionCrc16	1000	7,119.87 ns	139.513 ns	229.224 ns	0.0305	304 B

Figura 49 Benchmark antes del cambio a Span<byte>. Fuente: propia.

Tras ejecutar el benchmark una cosa era clara: NullFXCRC16 es un orden de magnitud más rápido que la función CRC de System.Data.HashFunction.CRC. Sin embargo, la API con la que se pueden usar las herramientas de NullFX es más arcaica y no acepta spans, cosa imprescindible en este proyecto.

Para solventar este problema, se ha realizado un fork al paquete NullFX, y se ha modificado el código fuente para poder utilizar spans en vez de arrays.

Este cambio, que en principio no debería suponer un cambio significativo en el rendimiento, acabó aumentando ligeramente los tiempos de ejecución en los casos con el número de bytes más bajo. Sin embargo, aligeró ligeramente los tiempos de ejecución de las iteraciones con el número de bytes más alto. Teniendo en cuenta que el tamaño mínimo que se va a utilizar no debería situarse por debajo de los 50 bytes, y que el tamaño máximo puede aumentar hasta los 1500 (con esta MTU), se puede afirmar que el rendimiento de la versión que usa spans es ligeramente superior a la versión inicial, aunque en la mayoría de los casos se encuentre suficientemente cerca de la desviación estándar como para que el incremento no sea considerable.

Method	ArraySize	Mean	Error	StdDev	Gen 0	Allocated
NullFxCrc16	10	15.71 ns	0.176 ns	0.156 ns	-	-
DataHashFunctionCrc16	10	242.70 ns	4.680 ns	4.597 ns	0.0362	304 B
NullFxCrc16	100	199.76 ns	3.928 ns	4.203 ns	-	-
DataHashFunctionCrc16	100	820.82 ns	16.258 ns	15.967 ns	0.0362	304 B
NullFxCrc16	1000	2,086.71 ns	30.372 ns	25.362 ns	-	-
DataHashFunctionCrc16	1000	6,809.31 ns	135.442 ns	120.065 ns	0.0305	304 B

Figura 50 Benchmark después del cambio a Span<byte>. Fuente: propia.

7.1.5 Otros protocolos residentes en el controlador

Para garantizar un correcto funcionamiento de la pila de red IPDP se requiere que el controlador IPDP ofrezca ciertos servicios de capas superiores, como puede ser DNDP, SAP, y MCP.

DNDP es necesario en este nivel para detectar e informar de las redes IPDP disponibles. El fragmento de código que se encarga de este protocolo reside tras la inicialización de los adaptadores de red, ya que se encarga de enviar un DNDP Discover a través de todos los adaptadores disponibles (y conectados).
En el loop principal del envío y recepción de paquetes, durante la recepción de un paquete, una parte del código se encarga de revisar si el paquete recibido es un paquete DNDP y si va destinado al socket interno del adaptador (el 0). Si este es el

caso, el controlador contesta a la petición con el subsecuente DNDP Advertise adecuado o adopta los parámetros recibidos a través de un DNDP Advertise.

En caso de que no se reciba un paquete DNDP Advertise tras la primera ronda de recepción de paquetes, el controlador se encarga de crear una dirección IPDP aleatoria y convertirla en la dirección IPDP de la nueva red creada.

Para obtener las cabeceras de los protocolos de capa 4 o superior se utiliza el siguiente método.

```
public static class DndpHeaderExtensions{
    4 usages  janlopera
    public static ref DndpHeader GetDndpHeader(this Span<byte> span)
    {
        if (span.Length > Unsafe.SizeOf<DndpHeader>())
        {
            span = span[..(Unsafe.SizeOf<DndpHeader>())];
        }
        return ref Unsafe.As<byte, DndpHeader>( source: ref MemoryMarshal.GetReference(span));
    }
}
```

Figura 51 Método para extraer una cabecera DNDP. Fuente: propia.

Este método, similar al resto de métodos que extraen cabeceras, tiene como input un span y, mediante el uso de la clase Unsafe, se hace un cast de la referencia del span a la referencia del header deseado y, tras eso, se retorna la referencia.

La implementación del protocolo SAP en el controlador consiste en la respuesta, dentro del loop de recepción, a los requests SAP. Estos requests deben ser respondidos mediante su respectivo NewService en caso de que en la lista de la tupla de SapServiceType y el socket exista una coincidencia con el servicio solicitado.

El proceso de respuesta de una solicitud SAP es similar al descrito anteriormente, pero con sus respectivos métodos para las cabeceras.

Gracias a la implementación del protocolo SAP se puede obtener una puerta de enlace para los adaptadores. Para este propósito se ha creado el servicio Gateway.d, al estilo de los servicios que se pueden encontrar en los routers pfSense, que busca una puerta de enlace predeterminada para cada adaptador de red mediante el uso de SAP.

La implementación del protocolo MCP en el controlador consiste en un pequeño fragmento de código para responder a peticiones de ECHO.

7.1.6 Memoria Compartida

Para poder transportar los datos de una aplicación hasta el controlador y viceversa, en un controlador soportado mediante el WDK (Windows Developer Kit), normalmente se hace uso de las funciones Read y Write. Estas funciones llaman a un objeto alojado en la memoria que es el encargado de proveer los datos.

En este caso, como no se dispone de estos privilegios, se ha tenido que emular el funcionamiento mediante Memory Mapped Files. Para ello, se ha creado una estructura que se encuentra en el inicio de cada Memory Mapped File. Esta estrcutura consta de los siguientes campos.

```
public struct SharedMemoryHeader
{
    public static readonly int BufferSize = 1514 + Unsafe.SizeOf<short>();
    public static readonly int MaxEntries = 100;


    public int OffsetUpperHalf;
    public int UpperHalf;
    public int LastReadUpperHalf;
    public int OffsetLowerHalf;
    public int LowerHalf;
    public int LastReadLowerHalf;
```

Figura 52 Cabecera de la memoria compartida. Fuente: propia.

Los campos que comienzan con Offset indican la cantidad de bytes, desde el puntero que apunta a la memoria compartida, hasta el inicio de sus respectivas regiones.

Se ha dividido la memoria en dos mitades, llamadas UpperHalf y LowerHalf, que forman los buffers de lectura y escritura entre la aplicación y el controlador.

Los campos UpperHalf y LowerHalf indican la posición al siguiente fragmento del buffer disponible para escribir. Cuando esta variable llega al final de la memoria compartida vuelve a su posición inicial, creando un buffer circular.

Los campos que empiezan por LastRead indican el último buffer leído por la aplicación. Es posible que, si la aplicación no lee los paquetes con suficiente rapidez, el buffer sobrescriba paquetes no leídos. Esto no resulta un problema, ya que el protocolo IPDP nunca ha prometido más garantías que el mismo “best effort” de IP.

De esta cabecera se exponen los siguientes métodos.

```
3 usages  janlopera
public static unsafe void PushUpper(SharedMemoryHeader* ptr, Span<byte> frame){...}

16 usages  janlopera
public static unsafe void PushLower(SharedMemoryHeader* ptr, Span<byte> frame){...}

janlopera
public static unsafe void PushLowerLower(SharedMemoryHeader* ptr, Span<byte> frame){...}

10 usages  janlopera
public static unsafe void PopUpper(SharedMemoryHeader* ptr, out Span<byte> frame){...}

1 usage  janlopera
public static unsafe void PopLower(SharedMemoryHeader* ptr, out Span<byte> frame){...}
```

Figura 53 Métodos para la gestión de memoria compartida. Fuente: propia.

Las operaciones de push son las encargadas de escribir en memoria. Todas requieren de un span que no incluya la cabecera ethernet, a diferencia del método PushLowerLower, que requiere de todo el frame entero.

Las operaciones de pop son las encargadas de extraer los frames de la memoria compartida.

La memoria compartida es uno de los pocos elementos en los que se requiere el uso de punteros en todo el código de la aplicación. Esto es debido a que se utiliza el puntero del Accesor a la Memory Mapped File para aumentar el rendimiento.

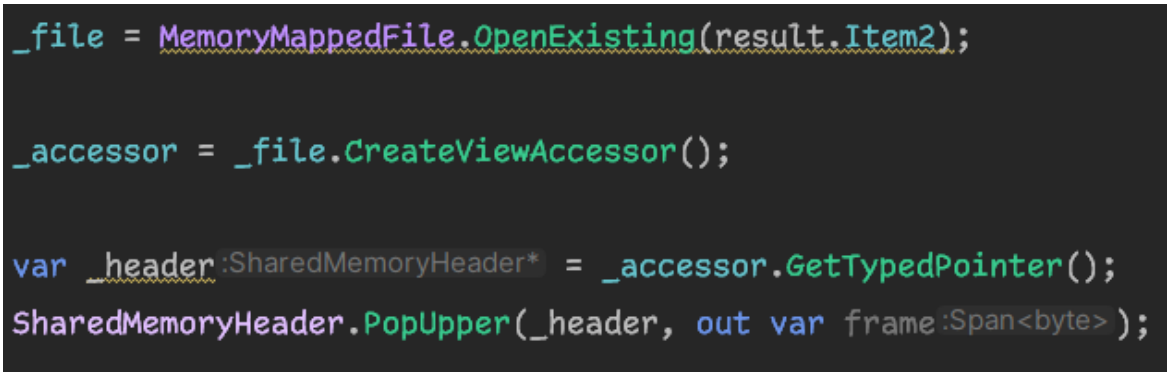


Figura 54 Procedimiento para obtener un el puntero a la memoria. Fuente: propia

7.1.7 Comunicación con el controlador

Las Memory Mapped Files son herramientas muy útiles para transferir grandes cantidades de información de forma continua entre aplicaciones, pero no son tan útiles para pequeños comandos entre aplicaciones; por lo que se ha optado por utilizar Pipes. Mediante la interfaz INetworkRequests se ha creado un mecanismo con el que las aplicaciones pueden comunicarse con el controlador.

```
public interface INetworkRequests
{
    2 usages  1 implementation  janlopera
    (bool success, string MemorySegment) RequestSocketUsage(ushort socket, byte protocol, int pid, byte gate = 0);
    2 usages  1 implementation  janlopera
    bool RequestSocketUnbind(ushort socket, byte protocol, int pid, byte gate = 0);
    1 usage  1 implementation  janlopera
    IpdpAddress[] RequestAddresses();
    1 implementation  janlopera
    bool AddSapService(ushort port, SapServiceTypeCode type);
    1 implementation  janlopera
    bool RemoveSapService(ushort port, SapServiceTypeCode type);
    1 usage  1 implementation  janlopera
    int GetNics();
    1 usage  1 implementation  janlopera
    ulong GetGateway(int nic);
    1 usage  1 implementation  janlopera
    void SetGateway(int nic, ulong gateway);
}
```

Figura 55 Interfaz INetworkRequests. Fuente: propia.

Mediante RequestSocketUsage y RequestSocketUnbind se crean las Memory Mapped Files correspondientes y se bindean los sockets con esta.

RequestAddresses es el método encargado de recuperar las direcciones IPDP de todos los NICs.

AddSapService Y RemoveSapService permiten añadir y eliminar servicios SAP a la lista de servicios del controlador.

GetNics, GetGateway y SetGateway permiten listar el número de NICs, obtener la puerta de enlace predeterminada y reemplazarla en caso de ser necesario.

Con estas operaciones, se completa la lista de requisitos necesarios para una correcta comunicación entre procesos.

7.2 IPDP Sockets (SDK)

Para que las aplicaciones se puedan comunicar mediante IPDP se ha creado un SDK que permite comunicarse con el controlador IPDP. Este SDK se conoce como IPDP Scokets.

La clase IpdpSocket es la encargada de encapsular toda la funcionalidad del SDK y busca ser similar al paquete de .Net System.Net.Sockets.

En el constructor de la clase se le especifica el tipo de socket y el protocolo. Por lo general, los sockets de tipo Datagrama funcionan mediante Pep, y los sockets de tipo Sequenced funcionan mediante Spep. Los sockets tipo Raw dan acceso de forma íntegra al paquete IPDP.

```
public IpdpSocket(SocketType type, IpdpProtocol protocol){...}
```

Figura 56 Constructor de la clase IpdpSocket. Fuente: propia.

El método Bind es el encargado de registrar con el controlador tanto el socket como el gate.
El método Connect tiene, de forma integrada, su propia llamada a Bind para poder recibir comunicaciones.

```
public bool Bind(ushort socketNumber, byte gateNumber = 0){...}
1 usage  janlopera
public bool Connect(IpdpAddress address, ushort socket, byte gate = 0){...}
```

Figura 57 Métodos Bind y Connect. Fuente: propia.

Los métodos Send y Receive son los encargados de enviar y recibir datos. Estos métodos pueden ser usados después de Bind o de Connect. En algunas ocasiones, como cuando se especifica la dirección destino, se puede usar sin Connect.

```
public unsafe (int Received, McpHeader? McpHeader, SpepHeader? SpepHeader) Receive(out byte[] data,
    out IpdpAddress senderAddress, out ushort senderSocket, out byte gate){...}

5 usages  janlopera *
public unsafe int Send(byte[] data, IpdpHeaderFlags flags = IpdpHeaderFlags.None,
    SpepFlags spepFlags = SpepFlags.None){...}

2 usages  janlopera
public unsafe int Send(byte[] data, IpdpAddress address, ushort socket, byte gate,
    IpdpHeaderFlags flags = IpdpHeaderFlags.None){...}
```

Figura 58 Métodos Send y Receive. Fuente: propia.

7.3 Pong Online

Para demostrar el funcionamiento de la suite de protocolos IPDP se ha creado una pequeña demo de un juego multijugador online: una versión del mítico pong, pero con multijugador. Para realizar la demo se ha decidido utilizar Unity, mostrando así que el SDK creado es compatible con el motor.

Para poder exportar el SDK a Unity es necesario compilarlo para .Net Standard 2.1, ya que es la versión más actualizada de .Net que el motor soporta por el momento, aunque hay planes para actualizarlo a .Net 6.0.

Esto implica que muchas de las APIs que se dan por supuesto en .Net 7 (o .Net 8) no están completamente disponibles en .Net Standard 2.1. La fundación dotNet ha creado backports de muchas de estas APIs mediante paquetes Nuget que son instalables y aportan prácticamente toda la compatibilidad de las APIs a las que reemplazan.

Al cambiar de versión objetivo e instalar los paquetes necesarios, ya se puede compilar el sdk para Unity.

Importar un managed dll en Unity es un proceso sencillo. Solo se debe depositar los dlls resultantes de la compilación en una carpeta dentro del directorio Assets y Unity automáticamente los incluye y linkea en tiempo de compilación.

Sin embargo, el proceso que a simple vista parecía sencillo (y debía serlo) resultó en incontables horas de debugging. Al hacer usar .Net Standard como objetivo de compilación, se espera que todas las aplicaciones compatibles con el estándar reaccionen de la misma forma, pero no es el caso de Unity.

Unity utiliza una versión modificada de Mono, un runtime alternativo de .Net, que es compatible con el estándar para la mayoría de los casos, pero en cuanto se profundiza lo suficiente, empiezan a haber divergencias en el código generado por el JIT Compiler. Estas divergencias llevan a que librerías que se ejecutan correctamente en el runtime de .net, pasen a dar desde NullReferenceExceptions hasta errores similares a una Segmentation Fault.

Para solventar este problema, la única solución posible es copiar el código del SDK y pegarlo dentro de una carpeta del proyecto, haciendo que el compilador de Unity sea el encargado de emitir el código IL.

Una vez solventado el problema, la programación del juego se ha resuelto con facilidad y sin ningún percance.

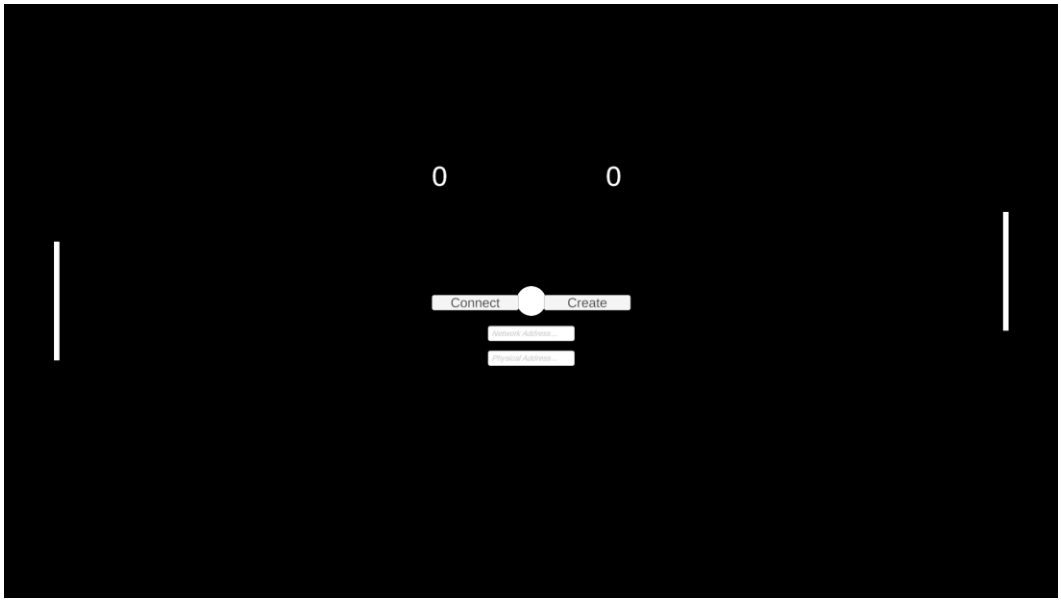


Figura 59 Captura ed la demo Pong. Fuente: propia.

7.4 Routing entre redes

Existen dos tipos de routing pensados para este prototipo del controlador. El primero es el routing simple implementado en el controlador y que se ejecuta cuando se usa el parámetro --route al ejecutar la aplicación. Este routing permite enrutar paquetes entre redes directamente conectadas y redes de las que ya se ha recibido algún paquete sin necesidad de hacer un Broadcast a la red.

Para lograr esto, se crea un diccionario compuesto por la primera mitad de una dirección IPDP y el NIC al que debe ser enviado el paquete si se desea llegar a esa red.

El segundo routing es el conocido como “Routing in Upper Layer”. Este método pasa las tramas de red de forma íntegra a una aplicación registrada en el socket 0. Esta aplicación debe ser la encargada de hacer el routing e implementar el resto de los protocolos, como RRP y RIEP, que se escapan del alcance del prototipo.

Este método de routing está pensado para que se creen aplicaciones y/o sistemas operativos complejos pensados para routing al estilo de pfSense o OpenWrt.

7.5 Resultados

Desde un primer momento el rendimiento ha sido una prioridad al plantear la programación del controlador; sin embargo, hasta que no se ha tenido construida la capa 4 de la suite de protocolos ha sido imposible obtener una medida real del rendimiento.

Durante el desarrollo se ha ido haciendo mediciones mediante los programas dotMemory y dotTrace de Jetbrains. Con estos programas se ha podido obtener una idea de en qué lugares se producían más memory allocations y en qué secciones del código el programa tardaba más en ejecutarse.

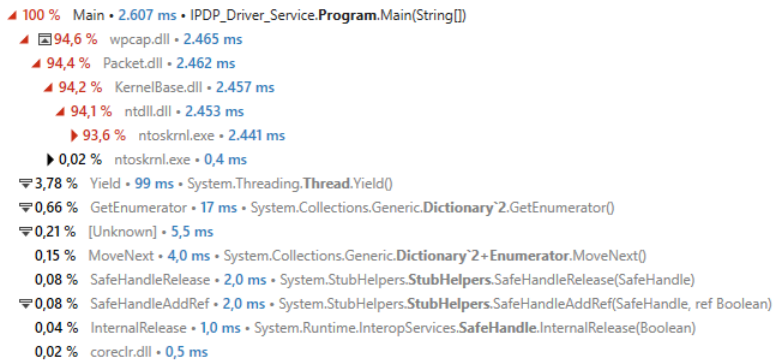


Figura 60 Calltree de dotTrace. Fuente: propia.

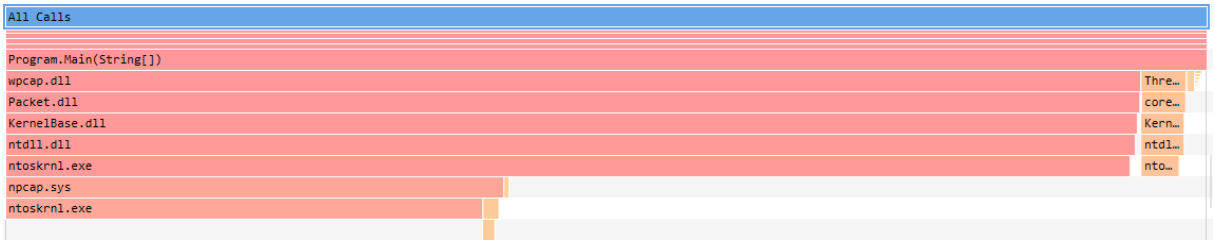


Figura 61 Flame Graph de dotTrace. Fuente: propia.

Gracias a estos programas, y a la imposibilidad de sacar más métricas de rendimiento en esos momentos del desarrollo, se ha conseguido optimizar el controlador para que la llamada más costosa sea la llamada al unmanaged dll.

En la mayoría de las publicaciones se habla del impacto negativo en el rendimiento que supone una llamada P/Invoke (una llamada a un unmanaged dll) (Gullberg, 2018); por lo que desde un primer momento se pensó que el factor limitante que impediría tener un buen rendimiento sería este procedimiento. Debido a esto, se estuvo planteando utilizar una de las funciones más escondidas del runtime de .Net: la emisión de código IL.

La intención de emitir código IL es la de realizar las llamadas a las funciones de la librería de forma manual: cargando los argumentos de la función, la dirección de memoria donde reside la función en el DLL, y utilizando la instrucción calli (Call Indirect) para llamar a la función directamente y sin pasar por ningún tipo de control de seguridad por parte del runtime.

Sin embargo, se decidió esperar a tener los primeros resultados fiables de rendimiento para pasar a este tipo de optimizaciones.

```
PEP Ping received! 0.201ms
PEP Ping received! 0.197ms
PEP Ping received! 0.185ms
PEP Ping received! 0.191ms
PEP Ping received! 0.210ms
PEP Ping received! 0.204ms
PEP Ping received! 0.206ms
PEP Ping received! 0.200ms
PEP Ping received! 0.215ms
PEP Ping received! 0.195ms
```

Figura 62 Ping realizado sobre el protocolo PEP. Fuente: propia.

Los primeros resultados fueron un ping (utilizando PEP y no MCP) entre dos dispositivos de la misma red situados a solo un switch de distancia. Ambos dispositivos cuentan con un procesador de 64 bits y están utilizando la misma versión de la versión preliminar de .Net 8.

Cuando se obtuvo una cifra inferior al milisegundo como RTT, se decidió medir el rendimiento contra el stack tradicional de Windows 11. La pila de red TCP/IP de Windows arrojó los siguientes resultados en un ping ICMP.

```
From 192.168.1.39: bytes=60 seq=000d TTL=128 ID=3f88 time=0.670ms
From 192.168.1.39: bytes=60 seq=000e TTL=128 ID=3f89 time=0.658ms
From 192.168.1.39: bytes=60 seq=000f TTL=128 ID=3f8a time=0.701ms
From 192.168.1.39: bytes=60 seq=0010 TTL=128 ID=3f8b time=5.740ms
From 192.168.1.39: bytes=60 seq=0011 TTL=128 ID=3f8c time=0.629ms
From 192.168.1.39: bytes=60 seq=0012 TTL=128 ID=3f8f time=0.707ms
```

Figura 63 Ping realizado sobre ICMP. Fuente: propia.

Con estos resultados, se puede afirmar que el stack de red IPDP, en su estado actual, es de dos a tres veces más rápido que la pila de red TCP/IP en Windows.

Además de conseguir un resultado, en cuanto a rendimiento, remarcable; también se ha obtenido un muy buen resultado en cuanto a utilización de memoria. dotMemory muestra un uso estable de memoria y sin recolecciones de basura, por lo que la latencia del controlador es estable y no varía significativamente con el paso del tiempo, al igual que también demuestra que no hay ninguna fuga de memoria en la aplicación.

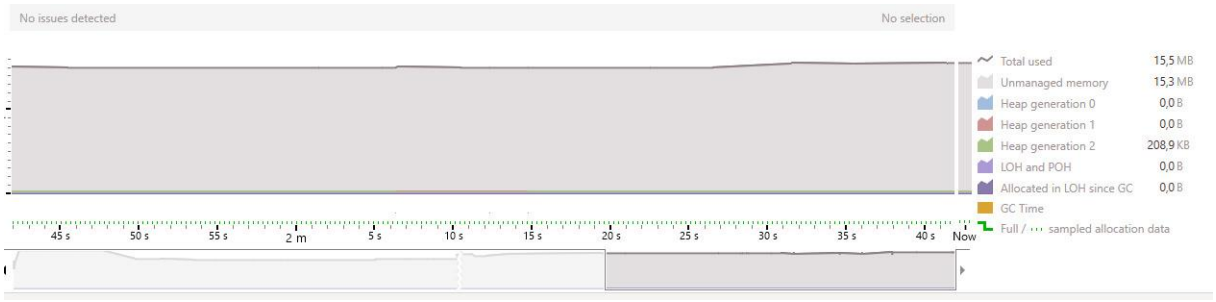


Figura 64 Captura de dotMemory mostrando las generaciones del recolector de basura. Fuente: propia.

Capítulo VIII: Conclusiones

Este trabajo de fin de grado ha estudiado la posibilidad del uso de una pila de protocolos diferente a la pila TCP/IP tradicional, poniendo en marcha una pila alternativa con unos protocolos que ofrecen una funcionalidad similar y, en algunos casos, superior a la ofrecida por el stack tradicional.

En el contexto y los antecedentes se ha estudiado, de forma breve, la historia de las redes; desde las redes de paquetes conmutados hasta el establecimiento de IP como el estándar que es hoy en día.

Durante el marco teórico se ha estudiado las capas dos, tres, y cuatro del modelo OSI. Se ha entrado en profundidad en las cabeceras de los protocolos más comunes de cada protocolo, además de algún protocolo alternativo a los estándares. También se ha hecho hincapié en explicar cómo funciona el modelo OSI, los diferentes estándares de Ethernet, y las redes de paquetes conmutados.

La etapa de desarrollo se ha dividido en dos mitades: una mitad de diseño y la otra propiamente de desarrollo. En la etapa de diseño se ha diseñado una suite de protocolos especificando sus cabeceras, funcionamiento, y uso. Además, se ha tratado de diseñar los protocolos teniendo en cuenta las necesidades del gaming moderno en cuanto a latencia y jitter.

Al desarrollar la suite se ha hecho hincapié en conseguir un producto competitivo a nivel de rendimiento y en la creación de un SDK de bajo nivel sobre el que construir aplicaciones y juegos con componentes online multijugador.

Los objetivos planteados al inicio del proyecto se han logrado cumplir de forma satisfactoria; tanto los objetivos primarios como los secundarios han sido cumplidos, en algunos casos, con creces.

Tanto la filosofía escogida durante el desarrollo como las tecnologías utilizadas han demostrado ser eficaces para este tipo de proyectos pese a las limitaciones que aparentemente puedan tener.

La metodología escogida para este proyecto ha demostrado ser eficaz en todos los aspectos. La planificación inicial se ha conseguido cumplir, permitiendo así cumplir todos los objetivos y dedicarle una cantidad sustancial de tiempo a la investigación.

En conclusión, este proyecto ha mostrado la viabilidad de la creación de un protocolo que compita con el estándar y que puede significar una mejora en las latencias entre clientes de un juego multijugador. Además, se ha demostrado que la plataforma de .Net es eficaz a la hora de realizar programación de sistemas; siempre y cuando no se requiera el uso del WDK (a pesar de que hay un proyecto para portar las herramientas del Windows Driver Kit a la plataforma .Net).

Capítulo IX: Ampliaciones

Las posibles ampliaciones de este proyecto son muy abundantes, ya que el proyecto pretende ser la base sobre la que construir desde nuevos protocolos hasta juegos y aplicaciones.

Uno de los elementos en los que se debería dedicar más tiempo y mejorar es la programación del controlador, a la que se le podrían añadir los elementos no esenciales ya diseñados; como la fragmentación; la detección de MTU y, por ende, la variabilidad de la MTU a lo largo del stack; o el multithreading, permitiendo así aumentar aún más el rendimiento del controlador.

También se podrían implementar tanto RRP como RIEP, los protocolos diseñados para routing y alta disponibilidad de los servicios, en una plataforma de routing “on upper layer”. Esto permitiría competir con soluciones como OpenWrt y pfSense si además se implementan medidas de control de acceso y firewall.

Otra forma de ampliar el proyecto es la creación de más protocolos que complementen y aumenten la funcionalidad de los ya existentes; desde un rediseño de H.323 hasta la creación de un protocolo del estilo DNS diseñado exclusivamente para esta suite.

Por último, se podría ampliar el SDK y exponer APIs similares a las de TcpClient y UdpClient de .Net, además de otros estilos de APIs. También se podrían crear las variaciones Async de los métodos usados para garantizar un uso correcto de los recursos y aumentar el rendimiento de las aplicaciones además de su capacidad de reacción. También sería posible exportar el SDK como un DLL clásico (mediante NativeAOT), permitiendo así que aplicaciones que soportan la cdecl (Calling Conventions de C) puedan hacer llamadas a la librería.

9.1 Escalado

En el supuesto caso de que el éxito de esta suite de protocolos, y por ende de esta pila de red, sea muy elevado es importante plantear una estrategia de despliegue e implantación gradual en la red.

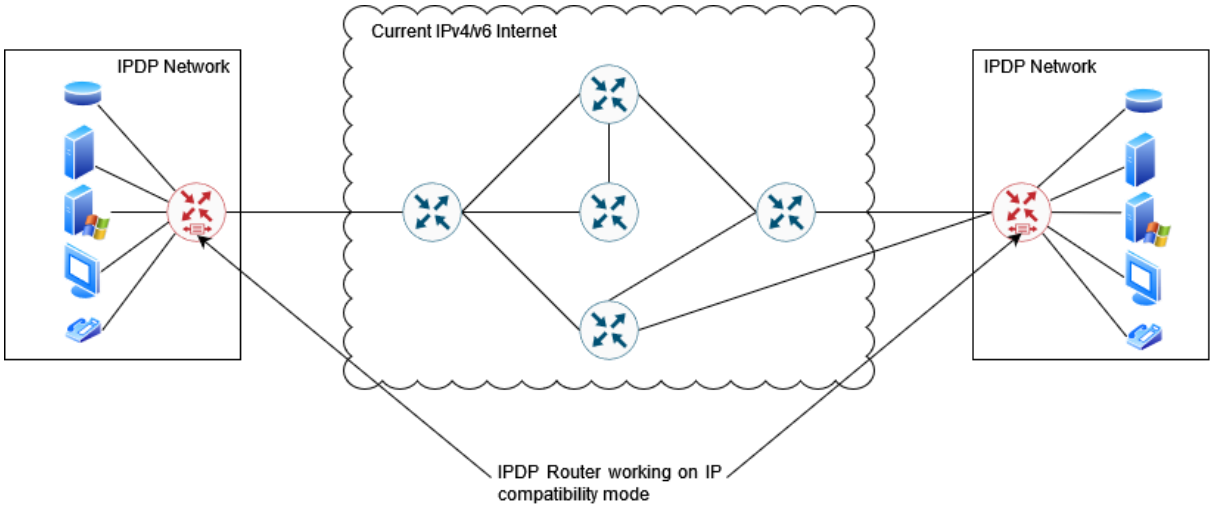


Figura 65 Despliegue inicial de IPDP. Fuente: propia.

En la primera fase de despliegue lo que se espera es que sean las redes domésticas y empresariales las que empiecen a dar soporte a IPDP. Durante esta primera fase, se reservan las direcciones de red IPDP (la fracción de red de una dirección IPDP) de la 00-00-00-00-00-00-00-00 a la 00-00-00-00-FF-FF-FF-FF. De esta forma, se asegura la conectividad con las redes IPv4, que conforman la mayor parte de las direcciones de internet mediante el modo CMP. En caso de que se quiera acceder a IPv6 desde IPDP entra en juego uno de los flags de la cabecera IPDP: EXT.

Con el flag EXT se habilitan las direcciones de red IPDP de 128 bits, permitiendo así encapsular toda una dirección IPv6 en ese segmento del paquete. Con el flag CMP se habilita el modo compatibilidad que permite la conversión y el enrutado entre protocolos diferentes.

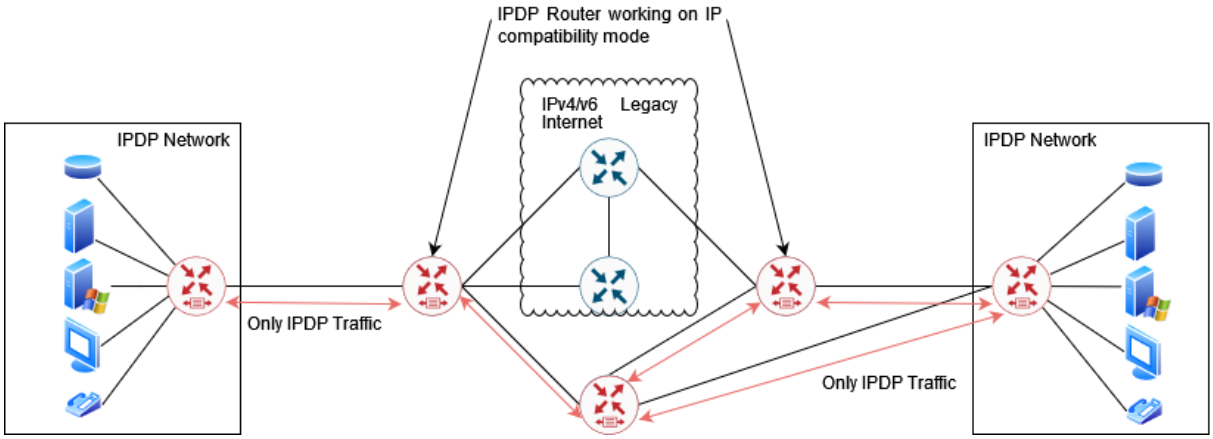


Figura 66 Segunda fase del despliegue de IPDP. Fuente: propia.

En una segunda fase del hipotético escenario en el que IPDP decide ser implementado como el estándar de facto de Internet, se parte de la base de que las redes locales han ido migrando lentamente a este nuevo protocolo.

Teniendo gran parte de las redes en IPDP ya no es necesario que sean los routers de acceso a internet los que se encarguen de la conversión a IP; por lo que este modo queda relegado a los routers que conectan con una red IP únicamente. Las direcciones reservadas para IP todavía se mantienen reservadas, pero se empiezan a liberar las direcciones de clase C para su uso.

En esta fase, los grandes servicios de hosting e ISPs han realizado ya el cambio a IPDP, ya sea instalando una capa de compatibilidad en su hardware existente para poder utilizar todo tipo de tráfico o migrando completamente a IPDP; por lo que los únicos nodos IP que quedan son de pequeños operadores, servicios viejos que no pueden ser actualizados lentamente, o redes no mantenidas.

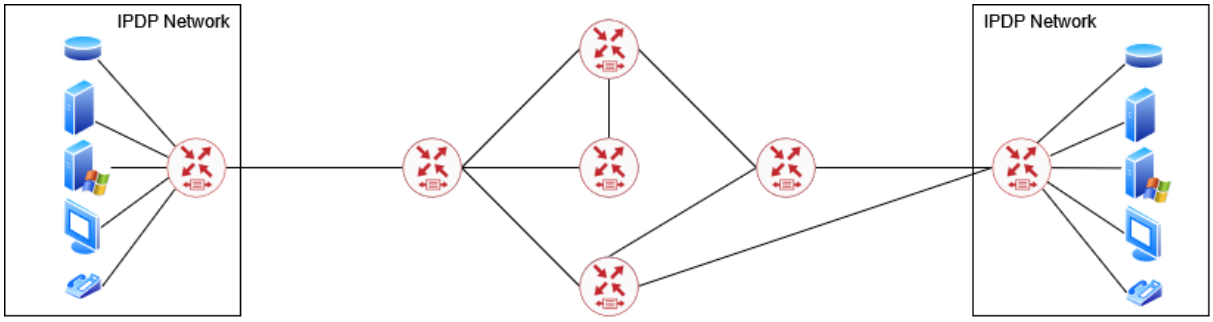


Figura 67 Fase final del despliegue de IPDP. Fuente: propia.

En la fase final de implementación de IPDP se parte de la base de que ya no quedan nodos significativos del Legacy Internet y que los pocos servicios que funcionan en tecnologías tan antiguas que no pueden soportar otro stack de red utilizan algún tipo de software IP-over-IPDP para poder continuar con sus operaciones.

En esta fase, ya se han liberado todas las direcciones de red que habían sido reservadas para IP, por lo que el uso del modo EXT vuelve a estar limitado.

Los grandes proveedores de internet y servicios cloud han migrado toda su instalación a IPDP y se deshacen de las capas de compatibilidad. Los modos CMP se desactivan de forma generalizada y ya no hay ningún tipo de overhead por cambio de protocolo.

Con esta implementación escalonada se asegura una correcta transición y se mantiene la compatibilidad con los protocolos previos de forma sencilla y eficaz.

Capítulo X: Bibliografía

Alani, M. M. (2014). *Guide to OSI and TCP/IP Models* (1 ed.). Springer Cham. doi:10.1007/978-3-319-05152-9

Amante, S., Carpenter, B., Jiang, S., & Rajahalme, J. (2011). *IPv6 Flow Label Specification.* doi:10.17487/RFC6437

Arora, M. (2011). *The Art of Hardware Architecture.* New York: Springer.

Deering, S., & Hinden, R. (2017). *Internet Protocol, Version 6 (IPv6) Specification.* doi:10.17487/RFC8200

Duffy, J. (03 de Noviembre de 2015). *Blogging about Midori*. Obtenido de Joe Duffy's Blog: https://joeduffyblog.com/2015/11/03/blogging-about-midori/

Duffy, J. (3 de Noviembre de 2015). *Joe Duffy's Blog*. Obtenido de A Tale of Three Safeties: https://joeduffyblog.com/2015/11/03/a-tale-of-three-safeties/

Eastlake, D., & Zuniga, J. C. (22 de Febrero de 2022). *IANA: IEEE 802 Numbers*. Obtenido de IANA: https://www.iana.org/assignments/ieee-802-numbers/ieee-802-numbers.xhtml#ieee-802-numbers-1

Foley, M. J. (10 de Noviembre de 2015). *Whatever happened to Microsoft's Midori operating system project?* Obtenido de ZDNet: https://www.zdnet.com/article/whatever-happened-to-microsofts-midori-operating-system-project/

Fruhlinger, J. (22 de Marzo de 2022). ¿Qué es el IPv6 y por qué tarda tanto en adoptarse? *Computer World*.

Gullberg, J. (14 de Abril de 2018). *Medium: An Introduction to ADL (or how to double your native .NET interop performance)*. Obtenido de Medium: https://medium.com/@jarl.gullberg/an-introduction-to-adl-or-how-to-double-your-native-net-interop-performance-c008e4da54db

IEEE International Standard for Information technology. (1998). IEEE International Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 2: Logical Link Control., (págs. 1-256).

IEEE International Standard for Information Technology. (2002). IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture. *IEEE Std*

802-2001 (Revision of IEEE Std 802-1990), 1-48. doi:10.1109/IEEESTD.2002.93395

Information Sciences Institute University of Southern California. (1981). *Internet Protocol: DARPA Internet Program Protocol Specification.* Marina del Rey. doi:10.17487/RFC0791

Information Sciences Institute University of Southern California. (1981). *TRANSMISSION CONTROL PROTOCOL: DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION.* Marina del Rey. doi:10.17487/RFC0793

Kaur, S., Singh, K., & Singh, Y. (2016). A comparative analysis of unicast, multicast, broadcast and Anycast addressing schemes routing in MANETs. *International Journal of Computer Applications, 133*(9), 16-22.

Kessler, G. C. (2004). An overview of TCP/IP protocols and the internet. *InterNIC Document*.

Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., . . . Wolff, S. (2009, Octubre). A Brief History of the Internet. *Computer Communication Review*, pp. 22-31. doi:10.1145/1629607.1629613

Microsoft. (15 de Septiembre de 2021). *Microsoft Learn: What is "managed code"?* Obtenido de What is "managed code"?: https://learn.microsoft.com/en-us/dotnet/standard/managed-code

Microsoft. (28 de Febrero de 2023). *Microsoft Learn: Fundamentals of garbage collection*. Obtenido de Microsoft Learn: https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals

Novell. (2000). *Novell Internet Access Server 4.1 Routing Concepts: The IPX Protocol.* Obtenido de Novell Internet Access Server 4.1 Routing Concepts: https://web.archive.org/web/20090528021242/http://www.novell.com/documentation/nw5/docui/index.html#../uscomm/rtcn_enu/data/hofxxkto.html

Novell. (2001). *Novell NetWare 6: Internetwork Packet Exchange.* Provo: Novell, Inc.

Postel, J. (1980). *User Datagram Protocol.* RFC. doi:10.17487/RFC0768

Postel, J., & Reynolds, J. (Febrero de 1988). A Standard for the Transmission of IP Datagrams over IEEE 802 Networks. doi:10.17487/RFC1042

Provan, D. (17 de Septiembre de 1993). *Ethernet Frame Types: Provan's Definitive Answer*. Recuperado el 09 de Enero de 2023, de

https://web.archive.org/web/20160529201807/http://www.ee.siue.edu/~bnoble/comp/networks/frametypes.html

Revich, Y., & Shilov, V. (10 de Marzo de 2017). Soviet Computer Technology in Non-Public Assessments of Contemporaries. *2017 Fourth International Conference on Computer Technology in Russia and in the Former Soviet Union*, págs. 1-6. doi:10.1109/SoRuCom.2017.00038

SDXCentral. (02 de Febrero de 2023). *What is a Circuit-Switched Network*. Obtenido de Circuit-Switched Network: https://www.sdxcentral.com/resources/glossary/circuit-switched-network/

Spurgeon, C. E. (2000). *Ethernet: The Definitive Guide.* O'Reilly Media, Inc.

Stadlmeier, M. (2018). *Writing Network Drivers in C#.* Munich: TECHNICAL UNIVERSITY OF MUNICH, DEPARTMENT OF INFORMATICS.

Ter-Gazarian, A. (27 de Septiembre de 2014). La malograda historia de la informática en la URSS. *Russia Beyond*.

Ullah, D. Z. (2012). Use of Ethernet Technology in Computer Network. *Global Journal of Computer Science and Technology*, 37–39.

W. Eddy, E. (2022). *Transmission Control Protocol (TCP).* doi:10.17487/RFC9293

Waitzman, D. (1 de Abril de 1990). Standard for the transmission of IP datagrams on avian carriers. doi:10.17487/RFC1149

Wong, R. (2018). *Mastering Recerse Engineering.* Birmingham: Packt Publishing Ltd.