UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Configuración y Uso de MPTCP en Dispositivos Móviles

Autora

Alejandra Oliver Boada

Directores

Jorge Navarro Ortiz, Félix Delgado Ferro

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Julio de 2023

Configuración y Uso de MPTCP en Dispositivos Móviles

Autora
Alejandra Oliver Boada

Directores
Jorge Navarro Ortiz, Félix Delgado Ferro

Configuración y Uso de MPTCP en Dispositivos Móviles

Alejandra Oliver Boada

Palabras clave: MPTCP, flujo, subflujo, *scheduler*, *path manager*, *kernel*, *mptcpd*, *routing*, eBPF, *throughput*, retardo.

Resumen

En la actualidad tecnológica en la que se encuentra el mundo, cada vez se hace un uso más extendido de dispositivos que se conectan a la red para comunicarse entre sí. Esta comunicación puede darse en forma cableada o más recientemente en forma inalámbrica; sin embargo, solo es posible utilizar una interfaz simultáneamente para ello, ya que la manera habitual de establecer una comunicación es mediante una de las múltiples interfaces del dispositivo que quiere realizar la conexión. El protocolo básico y por excelencia usado en las redes de comunicaciones es *Transmission Control Protocol* (TCP) y en un principio era una buena base, pero con la evolución tecnológica que se ha orquestado en los últimos años, el volumen de tráfico que discurre por las redes es tal que en muchos casos no es posible abordarlo de una manera eficiente y optimizada.

El problema del protocolo TCP reside en que no ha evolucionado a la par que las redes de comunicaciones y por ello, cada vez se tiene más problema con la poca optimización que presenta. Además, al estar concebido para trabajar sobre entornos cableados, solventando el problema básico de estos que es la congestión, no posee las características necesarias para funcionar de forma correcta cuando se sitúa en entornos inalámbricos.

La solución a este problema lo dan las tecnologías multi-camino, en concreto, el protocolo *Multipath TCP* (MPTCP). Este protocolo, entendido como una extensión de TCP y actualmente en su segunda versión, es capaz de aprovechar las diferentes interfaces de un dispositivo para transmitir y recibir datos, creando así múltiples caminos, conexiones TCP o subflujos (como se quieran llamar), las cuales se engloban en una única conexión MPTCP general (flujo MPTCP) entre los participantes de una comunicación. Por ejemplo, un dispositivo capaz de transmitir y recibir datos por una interfaz mediante *Wireless Fidelity* (Wi-Fi), por otra mediante 5G, y por una última mediante *Light Fidelity* (Li-Fi), todo ello de forma simultánea.

Por un lado, esta solución aporta un mayor rendimiento y optimización a las redes al ser capaz de balancear el tráfico entre los diferentes caminos, amortiguando así el posible impacto que puede tener un gran volumen de tráfico.

Por otro lado, ofrece mayor robustez frente a situaciones adversas como la caída de alguna interfaz, pudiendo seguir enviando y recibiendo datos por las demás. Estos son algunos de los principales beneficios por los que MPTCP está ganando cada vez más terreno en el mundo de las comunicaciones.

En este trabajo se describe el diseño de dos redes virtuales en las que se usa el protocolo MPTCP para la comunicación entre dos dispositivos. Primero se diseña, implementa y configura una de las redes para montar un escenario con MPTCP en su versión inicial; después se realiza un nuevo diseño y puesta en marcha de la segunda red para albergar otro escenario, pero en este caso con MPTCP en su última versión. Por último, se presentan en detalle los experimentos y comprobaciones realizados sobre los sistemas virtuales que aseguran el buen funcionamiento de los mismos; así como la puesta en marcha de un escenario con equipos reales en los que se prueba el funcionamiento de MPTCP utilizando 5G y Wi-Fi. Con esto se pretenden mostrar las ventajas que trae el protocolo, y cuál es la situación de desarrollo actual en la que se encuentra.

MPTCP Configuration and Usage on Mobile Devices

Alejandra Oliver Boada

Keywords: MPTCP, flow, subflow, scheduler, path manager, kernel, mptcpd, routing, eBPF, throughput, delay.

Abstract

In today's technological world, there is an increasing use of devices that connect to the network to communicate with each other. This communication can be wired or more recently wireless; however, it is only possible to use one interface simultaneously for this purpose, as the usual way to establish communication is through one of the multiple interfaces of the device that wants to make the connection. The basic and quintessential protocol used in communication networks is *Transmission Control Protocol* (TCP) and was originally a good basis, but with the technological evolution that has been orchestrated in recent years, the volume of traffic flowing through networks is such that in many cases it is not possible to deal with it in an efficient and optimised way.

The problem with the TCP protocol lies in the fact that it has not evolved at the same time as communications networks and, therefore, there is an increasing problem with its lack of optimisation. Moreover, as it is designed to work in wired environments, solving the basic problem of these, which is congestion, it does not have the necessary characteristics to work correctly in wireless environments.

The solution to this problem is provided by multi-path technologies, specifically the *Multipath TCP* (MPTCP) protocol. This protocol, understood as an extension of TCP and currently in its second version, is capable of taking advantage of the different interfaces of a device to transmit and receive data, thus creating multiple paths, TCP connections or subflows (whatever you want to call it), which are included in a single general MPTCP connection (MPTCP flow) between the participants of a communication. For example, one device that is capable of transmit and receive data over one interface via *Wireless Fidelity* (Wi-Fi), another via 5G, and a final one via *Light Fidelity* (Li-Fi), all simultaneously.

On one hand, this solution brings greater performance and optimisation to networks by being able to balance traffic between the different paths, thus cushioning the potential impact of a large volume of traffic. On the other hand, it offers greater robustness against adverse situations such as

the failure of any interface, being able to continue sending and receiving data through the others. These are some of the main benefits why MPTCP is gaining more and more ground in the communications world.

This work describes the design of two virtual networks in which the MPTCP protocol is used for communication between two devices. First, one of the networks is designed, implemented and configured to set up a scenario with MPTCP in its initial version; then a new design and implementation of the second network is carried out to host another scenario, but in this case with MPTCP in its latest version. Finally, the experiments and tests carried out on the virtual systems, which guarantee their correct functioning, are presented in detail, as well as a implementation of a scenario with real equipment in which the operation of MPTCP using 5G and Wi-Fi is tested. This is intended to show the advantages that the protocol brings, and what is the current development situation in which it is.

Yo, **Alejandra Oliver Boada**, alumna de la titulación Grado en Tecnologías de Ingeniería de Telecomunicación de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77552695F, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: Alejandra Oliver Boada

Granada a 13 de Julio de 2023.

D. **Jorge Navarro Ortiz**, Profesor Titular de Universidad del Área de Ingeniería Telemática del Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada y

D. **Félix Delgado Ferro**, estudiante de doctorado en el Departamento Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada

Informan:

Que el presente trabajo, titulado ***Configuración y Uso de MPTCP en Dispositivos Móviles***, ha sido realizado bajo su supervisión por **Alejandra Oliver Boada**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 13 de julio de 2023.

Los directores:

Jorge Navarro Ortiz **Félix Delgado Ferro**

Agradecimientos

Tras estos 4 años de carrera, con momentos de alegría, tristeza, entusiasmo y en general, mucha intensidad, me quedo con las personas que han sido partícipes de ello. Todo este tiempo me ha ayudado a adquirir muchos conocimientos y a la vez he vivido experiencias inolvidables. Por ello, quiero dar las gracias:

A mis tutores, Jorge y Félix, por todo el apoyo recibido a lo largo de este proyecto, y por darme consejos y ayudarme en todo lo posible. Me he sentido muy respaldada y me han cargado de ganas siempre.

A mis padres, Luis y Sandra, y a mi hermano Álvaro, por estar ahí en cada momento vivido, por aguantar mis agobios y celebrar conmigo mis logros. Gracias por apoyarme e impulsarme siempre en mis metas, y hacer que hoy sea la persona que soy y haya llegado donde estoy.

A mis amigas, Estrella, Ángela y Aida, y a Alejandro, por hacer de estos cuatro años los mejores vividos, y por habernos apoyado entre nosotros con tranquilidad y consejos en los peores momentos.

Y a todos mis compañeros, gracias por haber contribuido en algún momento de una u otra manera. Hemos formado un grupo muy valioso.

Gracias a todos de corazón.

Índice general

Glosario **21**

1. Introducción **27**
1.1. Contexto y Motivación 27
1.2. Objetivos y Alcance del Proyecto 29
1.2.1. Objetivos 29
1.2.2. Alcance del Proyecto 30
1.3. Metodología 30
1.4. Estructura de la Memoria 31

2. Estado del Arte **35**
2.1. Casos de Uso 35
2.2. Proyectos Relevantes 36
2.2.1. MPTCP en la Industria 4.0 39
2.3. Implementaciones Disponibles 41

3. Fundamentos Teóricos **43**
3.1. Protocolo TCP 43
3.2. Protocolo MPTCP 48
3.2.1. Conceptos Fundamentales 49
3.2.2. Requisitos de Diseño 49
3.2.3. MPTCP Versión 0 50
3.2.4. MPTCP Versión 1 68
3.3. MPTCP en el *Kernel* de Linux 76
3.3.1. Plan de Trabajo en Desarrollo 77

4. Planificación y Costes **79**
4.1. Planificación Temporal 79
4.2. Recursos y Coste 82
4.2.1. Recursos Humanos 82
4.2.2. Recursos Hardware 84
4.2.3. Recursos Software 84
4.2.4. Estimación de Costes 87
4.2.5. Modificaciones en la Planificación Inicial 88

5. Diseño e Implementación **89**
5.1. Contexto Práctico 89
5.2. Marco de Configuración e Implementación de MPTCP Versión 0 91
5.2.1. Estudio de Soluciones 91
5.2.2. Diseño de la Red 93
5.2.3. Creación del Escenario mediante *Vagrant* 94
5.3. Desarrollo del Entorno Práctico para MPTCP Versión 1 . . . 102
5.3.1. Repaso de Alternativas 102
5.3.2. Diseño del Escenario 104
5.3.3. Creación del Escenario para Albergar MPTCP Versión 1 en la Red 105

6. Pruebas y Experimentación **117**
6.1. Estudio de la Señalización del Protocolo 118
6.2. Montaje y Configuración del Entorno de Pruebas 120
6.3. *Throughput* en MPTCP Versión 0 122
6.3.1. Escenario Real con 5G y Wi-Fi 130
6.4. *Throughput* en MPTCP Versión 1 135
6.5. Complicaciones Encontradas en el Proyecto 140

7. Conclusiones y Líneas Futuras **143**
7.1. Conclusiones 143
7.2. Líneas Futuras 145

A. Archivos Necesarios en la Ejecución de Vagrant **159**

B. Configuración del *Routing* en el Escenario con MPTCP Versión 1 **165**

C. *Script* de *Matrix Laboratory (MatLab)* para la Obtención de la *Cumulative Distribution Function* (CDF) de los *Schedulers* **167**

Índice de figuras

1.1. Evolución del Uso de MPTCP [4]. 28
1.2. Conexiones MPTCP a Nivel Mundial en 2017 [5]. 29

2.1. Objetivo de *Multipath Quick UDP Internet Connections* (MP-QUIC) [18]. 37
2.2. Interfaz de Usuario de MPTCP App [20]. 37
2.3. MPTCP en el *Cloud - Augmented Multipath TCP* (A-MPTCP) [23]. 38
2.4. MPTCP en Redes Vehiculares [25]. 39
2.5. Arquitectura del Sistema 5G-CLARITY [26]. 40

3.1. Cabecera TCP [34]. 43
3.2. TCP-Tahoe vs. TCP-Reno. 45
3.3. Establecimiento de Conexión TCP. 47
3.4. Cierre de Conexión TCP. 47
3.5. MPTCP en el Campo de Opciones de TCP [39]. 48
3.6. MPTCP en el Modelo *Transmission Control Protocol/Internet Protocol* (TCP/IP) [40]. 48
3.7. Visión General del Protocolo MPTCP [41]. 49
3.8. Opción MP_CAPABLE en MPTCP Versión 0 [2]. 51
3.9. Establecimiento de Conexión en MPTCP Versión 0. 52
3.10. Opción MP_JOIN en MPTCP Versión 0 [2]. 52
3.11. Adición de un Nuevo Subflujo en MPTCP Versión 0. 54
3.12. Anuncio de una Nueva Dirección en MPTCP Versión 0. . . . 55
3.13. Opción ADD_ADDR en MPTCP Versión 0 [2]. 55
3.14. Opción REMOVE_ADDR en MPTCP Versión 0 [2]. 56
3.15. Eliminación de una Nueva Dirección en MPTCP Versión 0. . 56
3.16. Opción *Data Sequence Signal* (DSS) en MPTCP Versión 0 [2]. 56
3.17. Niveles de Jerarquía en la Transmisión de Datos en MPTCP Versión 0. 58
3.18. Pseudo encabezado para *Checksum* en MPTCP Versión 0 [2]. 59
3.19. Opción MP_PRIO en MPTCP Versión 0 [2]. 61
3.20. Opción MP_FAIL en MPTCP Versión 0 [2]. 62

3.21. Opción MP_FASTCLOSE en MPTCP Versión 0 [2]. 62
3.22. Cierre Rápido en MPTCP Versión 0. 63
3.23. *Middleboxes* Eliminando Opción MP_CAPABLE en MPTCP Versión 0 [2]. 66
3.24. Opción MP_CAPABLE en MPTCP Versión 1 [3]. 71
3.25. Caso 1 de Establecimiento de Conexión en MPTCP Versión 1. 71
3.26. Caso 2 de Establecimiento de Conexión en MPTCP Versión 1. 72
3.27. Caso 3 de Establecimiento de Conexión en MPTCP Versión 1. 72
3.28. Opción ADD_ADDR en MPTCP Versión 1 [3]. 73
3.29. Anuncio de una Nueva Dirección en MPTCP Versión 1. . . . 74
3.30. Opción MP_PRIO en MPTCP Versión 1 [3]. 75
3.31. Opción MP_TCPRST en MPTCP Versión 1 [3]. 75

4.1. Diagrama de Gantt. 81
4.2. Resumen de la Temporización de las Fases y Tareas del Proyecto. 82

5.1. Entorno Práctico con MPTCP en el *Kernel* de Linux [81]. . . 89
5.2. Escenario Virtual para MPTCP Versión 0. 93
5.3. Conexión *Secure Shell* (SSH) a *mptcpUe1* y *mptcpUe2*. . . . 95
5.4. Configuración del *Routing* en Ambas Máquinas Virtuales (MPTCP versión 0). 99
5.5. Configuración de Variables *sysctl net.mptcp*. 99
5.6. Prueba Inicial de MPTCP Versión 0. 101
5.7. Subflujos de Reserva en una Conexión con MPTCP Versión 0. 102
5.8. Escenario Virtual para MPTCP Versión 1. 104
5.9. Configuración del *Routing* en Ambas Máquinas Virtuales (MPTCP versión 1). 107
5.10. Configuración del *Path Manager* en el Cliente. 109
5.11. Configuración del *Path Manager* en el Servidor. 109
5.12. Configuración del *Path Manager* en Ambas Máquinas Virtuales (MPTCP versión 1). 109
5.13. Comprobación de la Compilación y Carga de los *Schedulers* en el *Kernel*. 112
5.14. Variables *sysctl* en MPTCP Versión 1. 113
5.15. Prueba Inicial de MPTCP Versión 1. 114
5.16. Subflujos de Reserva en una Conexión con MPTCP Versión 1. 115

6.1. Intercambio MP_CAPABLE. 118
6.2. Mensajes ADD_ADDR y DSS. 119
6.3. Intercambio MP_JOIN. 119
6.4. Intercambio FIN. 120
6.5. Limitación de Velocidad en las Interfaces de las Máquinas. . . 123
6.6. Test con *Iperf3* en el Servidor (*mptcpUe1*). 123

6.7. Test con *Iperf3* en el Cliente (*mptcpUe2*). 123
6.8. Comportamiento del *Default Scheduler* en MPTCP Versión 0. 124
6.9. Comportamiento del *Default Scheduler* con Retardos Añadidos en MPTCP Versión 0. 125
6.10. Comportamiento del *Default Scheduler* sin Limitación de *Throughput* y con Retardos Añadidos en MPTCP Versión 0. . . 125
6.11. Comportamiento del *Redundant Scheduler* en MPTCP Versión 0. 126
6.12. Comportamiento del *Redundant Scheduler* con Retardos Añadidos en MPTCP Versión 0. 126
6.13. Comportamiento del *Round-robin Scheduler* con CWND_LIMITED=Y en MPTCP Versión 0. 127
6.14. Comportamiento del *Round-robin Scheduler* con CWND_LIMITED=N en MPTCP Versión 0. 128
6.15. *Script* para el Procesamiento de los Datos Obtenidos en las Pruebas. 129
6.16. CDF de cada *Scheduler* en MPTCP Versión 0. 129
6.17. Esquema del Escenario MPTCP Versión 0 con 5G y Wi-Fi [96].130
6.18. Montaje del Escenario Real con 5G y Wi-Fi [96]. 131
6.19. Configuración del *Routing* y de MPTCP Versión 0 en el Servidor. 132
6.20. Configuración del *Routing* y de MPTCP Versión 0 en el Cliente.132
6.21. Puesta en Marcha de la Estación 5G. 133
6.22. Conexión del *Customer Provided Equipment* (CPE) a la Estación 5G. 133
6.23. *Graphical User Interface* (GUI) de Amarisoft. 134
6.24. Comportamiento de MPTCP Versión 0 con el *Round-robin Scheduler* y usando 5G y Wi-Fi. 134
6.25. Comportamiento del *Default Scheduler* en MPTCP Versión 1. 136
6.26. Comportamiento del *First Scheduler* en MPTCP Versión 1. . 137
6.27. Comportamiento del *Backup Scheduler* en MPTCP Versión 1. 137
6.28. Comportamiento del *Redundant Scheduler* en MPTCP Versión 1. 138
6.29. Comportamiento del *Redundant Scheduler* con Retardos Añadidos en MPTCP Versión 1. 139
6.30. CDF de cada *Scheduler* en MPTCP Versión 1. 139

A.1. Instalación del *Kernel* 5.4.144 y de *Openvpn*. 160
A.2. Instalación de Herramientas para MPTCP v0. 162
A.3. Clonación de *multitechnology_testbed_v0* en las Máquinas. . . 162
A.4. Archivo de Configuración *Vagrantfile* utilizado por *Vagrant* . 163

B.1. *Routing* en el Cliente. 165
B.2. *Routing* en el Servidor. 166

C.1. Obtención de la CDF de cada *Scheduler* (Parte 1). 167
C.2. Obtención de la CDF de cada *Scheduler* (Parte 2). 168

Índice de Tablas

4.1. Tabla Resumen de Horas Dedicadas a las Diferentes Tareas. . 83
4.2. Coste Total de Recursos Humanos. 84
4.3. Características del Portátil Utilizado. 84
4.4. Coste Total de Recursos Hardware. 84
4.5. Coste Total de Recursos Software. 87
4.6. Coste Total del Proyecto. 88
4.7. Coste Total de los Equipos del Escenario Real. 88

Glosario

3GPP R16 *3rd Generation Partnership Project Release 16*.

5G NR *5G New Radio*.

A-MPTCP *Augmented Multipath TCP*.

ACK *Acknowledgment*.

AMD *Advanced Micro Devices*.

AP *Access Point*.

API *Application Programming Interface*.

ARQ *Automatic Repeat Request*.

BALIA *Balanced Linked Increase Algorithm*.

BPF *Berkeley Packet Filter*.

CDF *Cumulative Distribution Function*.

CentOS *Community ENTerprise Linux Operating System*.

CHNM *Center for History and New Media*.

CPD Centro de Procesamiento de Datos.

CPE *Customer Provided Equipment*.

CPU *Central Processing Unit*.

DHCP *Dynamic Host Configuration Protocol*.

DNS *Domain Name System*.

DNSSEC *Domain Name System Security Extensions*.

DoS *Denial Of Service*.

DSN *Data Sequence Number*.

DSS *Data Sequence Signal*.

eBPF *Extended Berkeley Packet Filter*.

EID *Endpoint Identifier*.

eMBB *Enhanced Mobile Broadband*.

FIN *Finalize*.

FreeBSD *Free Berkeley Software Distribution*.

GMU *George Mason University*.

gNB *gNodeB*.

GUI *Graphical User Interface*.

HMAC *Hash-based Message Authentication Code*.

i2CAT Internet e Innovación Digital en Cataluña.

IA Inteligencia Artificial.

IANA *Internet Assigned Numbers Authority*.

ID *Identification*.

IDS/IPS *Intrusion Detection System/Intrusion Prevention System*.

IDSN *Initial Data Sequence Number*.

IEEE *Institute of Electrical and Electronics Engineers*.

IETF *Internet Engineering Task Force*.

IHP *Innovations for High Performance Microelectronics*.

iOS *Iphone Operating System*.

IoT *Internet Of Things*.

IP *Internet Protocol*.

ISN *Initial Sequence Number*.

ISP *Internet Service Provider*.

KTLS *Linux Kernel Transport Layer Security.*

Li-Fi *Light Fidelity.*

LIA *Linked Increase Algorithm.*

LISP *Location/Identifier Separation Protocol.*

LPDDR4x *Low Power Double Data Rate 4x.*

LTE *Long Term Evolution.*

LTS *Long Term Support.*

MacOS *Macintosh Operating System.*

MAMI *Measurement and Arquitecture for a Middleboxed Internet.*

MatLab *Matrix Laboratory.*

MCP *Middlebox Cooperation Protocol.*

MNO *Mobile Network Operators.*

MPQUIC *Multipath Quick UDP Internet Connections.*

MPTCP *Multipath TCP.*

MSL *MaximunSegmentLifetime.*

multi-WAT *multi-Wireless Access Technologies.*

MySQL *My Structured Query Language.*

N3IWF *Non-3GPP Interworking Function.*

NAT *Network Address Translation.*

NFV *Network Function Virtualization.*

NixOS *Nix Operating System.*

NMVe *Non-Volatile Memory Express.*

NSA *Non-Stand Alone.*

NUC *Next Unit of Computing.*

OLIA *Oportunistic Linked Increase Algorithm.*

OpenSUSE *Open Software- und System-Entwicklung.*

OpenWRT *Open Wireless Router*.

PCIe *Peripheral Component Interconnect Express*.

PEP *Performance Enhancing Proxy*.

PSH *Push*.

PTO *Path Transparency Observatory*.

QoS *Quality Of Service*.

QUIC *Quick UDP Internet Connections*.

R16 *Release 16*.

RAM *Random Access Memory*.

RF Radio Frecuencia.

RFC *Request for Comments*.

RLOC *Routing Locator*.

RST *Reset*.

RTT *Round Trip Time*.

SA *Stand Alone*.

SACK *Selective Acknowledgment*.

SAN *Storage Area Network*.

SDN *Software Defined Networking*.

SHA1 *Secure Hash Algorithm 1*.

SHA256 *Secure Hash Algorithm 256*.

SN *Sequence Number*.

SO-DIMM *Small Outline Dual Inline Memory Module*.

SSD *Solid State Drive*.

SSH *Secure Shell*.

SSL *Secure Sockets Layer*.

SSN *Subflow Sequence Number*.

SYN *Synchronize*.

TCP *Transmission Control Protocol*.

TCP/IP *Transmission Control Protocol/Internet Protocol*.

TDD *Time Division Duplexing*.

TLS *Transport Layer Security*.

UDP *User Datagram Protocol*.

UGR Universidad de Granada.

ULP *Upper-Layer Protocol*.

URG *Urgent*.

URLLC *Ultra-Reliable & Low Latency Communications*.

VLAN *Virtual Local Area Network*.

VM *Virtual Machine*.

VPN *Virtual Private Network*.

Wi-Fi *Wireless Fidelity*.

WLAN *Wireless Local Area Network*.

wVegas *Delay based CC scheme tuned for MPTCP*.

Capítulo 1

Introducción

En el presente capítulo se pretende poner en contexto de la situación en la que se encuentra el protocolo *Multipath TCP* (MPTCP). Además, se comentan los objetivos perseguidos durante la realización del proyecto y se presenta de manera resumida y esquemática cómo se van a ir abordando los diferentes temas.

1.1. Contexto y Motivación

En las últimas décadas, se ha producido una gran evolución tecnológica en muy diversas áreas. Uno de los pilares que ha hecho esto posible es el gran avance en Internet y, en general, en las tecnologías de red. A principios de la década de 1960, con la primera investigación sobre conmutación de paquetes, comenzó el proceso de aparición de dispositivos interconectados; en la década de 1970 se desarrolló el protocolo *Transmission Control Protocol* (TCP). En ese entonces no se podía predecir la tasa de crecimiento que iban a tener las redes en todo el mundo.

Actualmente, se disponen de miles de millones de dispositivos interconectados, que junto con la era *Internet Of Things* (IoT) [1], hacen que cualquier equipo que se use en nuestra vida cotidiana tenga conectividad con muchos otros. Esta nueva realidad lleva a que el planteamiento de escenarios con dispositivos conectados a través de una única interfaz, ya sea vía *Mobile Network Operators* (MNO) o *Wireless Local Area Network* (WLAN), como vía *Wireless Fidelity* (Wi-Fi), creando rutas únicas punto a punto a través de TCP, resulte insuficiente, y se opte por la posibilidad de establecer múltiples rutas para la transmisión y recepción de datos entre equipos.

Esto, además de mejorar el rendimiento de la red, podría resolver problemas de conectividad; por ejemplo, si un equipo conectado a la red Wi-Fi pierde dicha conexión por los motivos que sean, rápidamente puede reaccionar utilizando otra de las redes (i.e: la red móvil).

Con el fin de dar sentido a este planteamiento, el grupo de trabajo del *Internet Engineering Task Force* (IETF) creó el protocolo MPTCP. En enero de 2013 se publicó su especificación como estándar experimental en el *Request for Comments* (RFC) 6824 [2], no fue hasta marzo de 2020 cuando surgió MPTCP versión 1 en el RFC 8684 [3].

Este protocolo amplía el TCP actual para introducir capacidades de múltiples rutas, manteniendo la retrocompatibilidad y evitando la interrupción de los *middleboxes*, como *proxies* (dispositivos intermediarios en la comunicación que realiza un dispositivo cliente a otro servidor).

Las principales ventajas que se buscan con el uso de MPTCP son:

1. **Incrementar la eficiencia**: es posible aumentar el ancho de banda de una conexión TCP al utilizar varias interfaces, con respecto al que se alcanzaría empleando una interfaz (un único camino).

2. **Aumentar la robustez de la conexión**: es posible retransmitir los datos por diversos caminos a la vez, reduciendo así la tasa de pérdidas y proporcionando estabilidad a la conexión. Por otro lado, usando un algoritmo de *scheduling* junto con un mecanismo de control de congestión, es posible distribuir el tráfico entre las diferentes interfaces según convenga.

Si se pone atención a la evolución de este, cada vez se ha ido generando más interés dentro de la comunidad científica, con cientos de artículos que hacen referencia a *Multipath TCP*. Esto se muestra en la figura 1.1, donde se observa el número acumulado de citas que tuvo este protocolo para la creación de borradores de Internet hasta 2018, los cuales posteriormente formaron el RFC 6824 [2]:

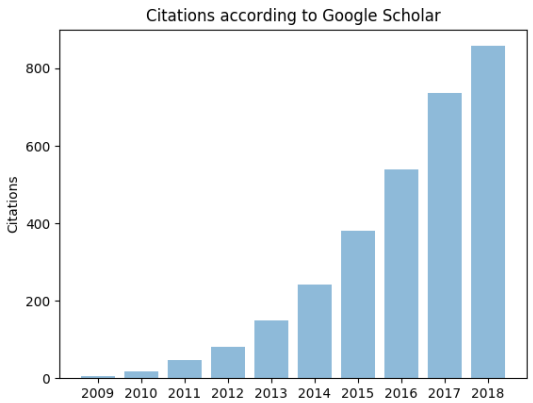


Figura 1.1: Evolución del Uso de MPTCP [4].

Además, el uso de MPTCP está siendo cada vez más demandado. Esto se puede apreciar en la figura 1.2, donde se muestra el número de conexiones MPTCP activas en el año 2017. Como cabe esperar, en la actualidad este número es mil veces superior.

Figura 1.2: Conexiones MPTCP a Nivel Mundial en 2017 [5].

Tras haber visualizado los gráficos anteriores, se puede observar como MPTCP ha sido ampliamente aceptado por la comunidad. Esto es así, ya que los estudios realizados pueden corroborar las diversas ventajas que posee este protocolo frente al TCP simple. En cuanto a su visión futura, los desarrollos e implementaciones cada vez son más completos y más detallados, incluyendo casi todas las funciones de las que dispone el protocolo. Aun así, todavía hay fallas y camino por recorrer para que MPTCP esté del todo estandarizado y haya implementaciones completas sin errores, ya que todavía hay entornos en desarrollo que no producen los resultados esperados. La actual ocupación de los investigadores es establecer una implementación de MPTCP completa en su versión final, que permita hacer pruebas mostrando las ventajas que se ofrecen.

1.2. Objetivos y Alcance del Proyecto

1.2.1. Objetivos

Los principales objetivos que se desean perseguir con la realización de este proyecto son:

1. Análisis de los principales artículos y contribuciones recientes relacionadas con el uso de MPTCP en redes de comunicaciones; tanto en entornos similares a los que se presentan en este proyecto, como en entornos más sofisticados.

2. Estudio de las principales características y funcionamiento de las dos versiones del protocolo (funciones más importantes y mensajes intercambiados).

3. Diseño, implementación y configuración de una primera red virtual con pleno funcionamiento de MPTCP versión 0.

4. Diseño, implementación y configuración de una segunda red virtual funcionando con MPTCP versión 1.

5. Realización de pruebas de funcionamiento, almacenamiento de información y creación de gráficas que ilustren las características del protocolo (trabajo de campo). Análisis comparativo del rendimiento en los múltiples casos.

1.2.2. Alcance del Proyecto

Las labores logradas a lo largo de este trabajo son:

- Presentación de los principales casos de uso que tiene MPTCP, junto con los proyectos más importantes llevados a cabo con este.

- Recopilación de información sobre el funcionamiento y las características del protocolo.

- Diseño e implementación de dos escenarios virtuales mediante diversas herramientas, con unas se busca tener funcionando correctamente MPTCP versión 0 y con otras, MPTCP en su versión más actual.

- Comprobación del funcionamiento del protocolo en los diferentes entornos creados, presentando diferentes situaciones (modificaciones en el ancho de banda). Además, creación de un entorno de pruebas real en el que se verifica el funcionamiento de MPTCP con 5G y Wi-Fi.

1.3. Metodología

En esta sección se van a explicar las diferentes tareas y/o procedimientos que se han ido llevando a cabo, ordenados temporalmente, y con los cuales se han conseguido alcanzar los objetivos propuestos.

En el presente proyecto se va a seguir una metodología en cascada, esta permite definir la estructura del trabajo de forma clara y facilitar su seguimiento (análisis, diseño, implementación, pruebas y observaciones, cada etapa se completa antes de pasar a la siguiente). Además, permite ir mostrando si se van consiguiendo los fines establecidos.

Lo primero que se ha realizado es mostrar un pequeño resumen introductorio, con el objetivo de ir enseñando al lector la situación de origen del protocolo. Después, se han definido los objetivos que se quieren alcanzar para poder tenerlos presentes a la hora de ir realizando cada tarea; posteriormente, se han presentado los principales casos de uso y proyectos recientemente realizados y/o que se están llevando a cabo actualmente en el ámbito MPTCP, además de las implementaciones más importantes que se tienen. Seguidamente, se han expuesto los conocimientos teóricos necesarios para poder comprender el protocolo y ofrecer una implementación correcta; además de mostrar el desarrollo temporal y coste económico que se ha ido teniendo en la elaboración de este proyecto.

El siguiente paso, una vez definidos los objetivos y expuesta la teoría necesaria y, la temporización y costes del estudio, es conveniente analizar las diferentes soluciones que sean capaces de adecuarse a los objetivos propuestos. Para ello, es importante plantearlas y evaluar los requisitos de cada una con el fin de elegir aquella que satisfaga nuestras necesidades. Una vez se ha elegido la solución que se va a implementar, hay que definir de forma clara su funcionamiento y seleccionar los componentes necesarios para el desarrollo de la red. Seguidamente, se procede a su diseño e implementación.

Por último, se lleva a cabo una fase de pruebas y experimentación que permita concluir si se cumplen los objetivos previstos y se corroboran los rendimientos esperados.

En una última tarea, se comentarán las conclusiones obtenidas y las posibles líneas de trabajos futuros.

1.4. Estructura de la Memoria

En esta última sección de este primer capítulo se exponen las diferentes partes de las que consta esta memoria, haciendo una breve descripción de cada una.

- **Capítulo 1. Introducción:** se exponen las siguientes partes a modo introductorio:
  - **Contexto y Motivación:** en este apartado se explican los motivos por los que surge este protocolo, así como sus principales beneficios y su evolución a lo largo de los años.
  - **Objetivos y Alcance del Proyecto:** se detallan los principales objetivos que se desean perseguir y los logros alcanzados.

- **Metodología:** se detalla de manera superficial el método de trabajo seguido, describiendo ligeramente los pasos que se han ido llevando a cabo en la elaboración del proyecto.
- **Estructura de la Memoria:** se describe el contenido principal de cada uno de los capítulos de los que consta esta memoria.

- **Capítulo 2. Estado del Arte:** este capítulo consta de 3 apartados:
  - **Casos de Uso:** se detallan los casos de uso más importantes junto con ejemplos de cada uno.
  - **Proyectos Relevantes:** se comentan proyectos en actual desarrollo o llevados a cabo en los últimos años que emplean MPTCP, junto con las tecnologías utilizadas en cada caso.
    - *MPTCP en la Industria 4.0*: se trata un proyecto en especial en el que participa la Universidad de Granada con MPTCP y tecnologías 4.0.
  - **Implementaciones Disponibles:** se describen las principales implementaciones que tiene el protocolo actualmente.

- **Capítulo 3. Fundamentos Teóricos:** se explican los conceptos teóricos necesarios para la comprensión de MPTCP, antes de pasar a la práctica. Se divide en 3 partes:
  - **Protocolo TCP:** se hace una introducción del protocolo TCP, el cual da paso a MPTCP.
  - **Protocolo MPTCP:** se introduce el protocolo y se divide en 4 partes.
    - *Conceptos Fundamentales:* descripción y situación de MPTCP en la pila *Transmission Control Protocol/Internet Protocol* (TCP/IP).
    - *Requisitos de Diseño:* premisas en la elaboración de MPTCP.
    - *MPTCP Versión 0:* características de la versión 0 del protocolo.
    - *MPTCP Versión 1:* características y funciones añadidas y/o mejoradas en la versión más actual del protocolo.
  - ***Kernel* de Linux:** se realiza una breve descripción de la implementación, ya que es la que se va a llevar a cabo en el proyecto.
    - *Plan de Trabajo en Desarrollo:* descripción de una serie de objetivos a largo plazo que se pretenden mejorar en ella.

- **Capítulo 4. Planificación y Costes:** se describen 3 partes:
  - **Planificación Temporal:** se realiza una descripción temporal de las tareas que se han ido llevando a cabo.

  - *Diagrama de Gantt:* se expone un diagrama en el que se observa el tiempo que ha llevado la realización de cada una de las tareas o fases en relación con el tiempo total que se tenía para todo el proyecto.
- **Recursos:** se muestran los recursos que se han necesitado.
  - *Recursos Humanos:* se enumeran los recursos humanos necesarios junto con su presupuesto total.
  - *Recursos Hardware:* se muestran los recursos hardware y su presupuesto total.
  - *Recursos Software:* se describen los recursos software empleados y su coste.
- **Estimación de Costes:** consiste en una estimación total de los costes del proyecto, suma de todos los recursos explotados.
- **Modificaciones en la Planificación Inicial:** explica los costes asociados a un nuevo escenario, en este caso real, cuyo planteamiento surge a mitad del proyecto.

- **Capítulo 5. Diseño e Implementación:** este capítulo describe los pasos seguidos para el diseño y puesta en marcha de las diferentes redes virtuales. Se divide en 3 partes:
  - **Contexto Práctico:** se presenta el entorno práctico que se va a crear de forma general.
  - **Marco de Configuración e Implementación de MPTCP Versión 0:** se explican los pasos seguidos para la puesta en marcha de un entorno virtual con funcionamiento del protocolo en versión 0 (elección de herramientas, diseño de la red y configuración de dispositivos). Todo ello dividido en 3 secciones:
    - *Estudio de Soluciones.*
    - *Diseño de la Red.*
    - *Creación del Escenario mediante Vagrant.*
  - **Desarrollo del Entorno Práctico para MPTCP Versión 1:** búsqueda de la mejor solución para poder implementar el protocolo en versión 1, diseño de una red que albergue el escenario, y configuración de esta. El proceso se describe en 3 partes:
    - *Repaso de Alternativas.*
    - *Diseño del Escenario.*
    - *Creación del Escenario para Albergar MPTCP Versión 1 en la Red.*

- **Capítulo 6. Pruebas y Experimentación:** se divide en 5 apartados:

- *Estudio de la Señalización:* se hace un estudio de la señalización del protocolo, con el fin de validar lo visto en teoría.
- *Montaje y Configuración del Entorno de Pruebas:* se describe el software y los elementos necesarios para la realización de las pruebas.
- *Throughput en MPTCP versión 0:* se presentan los tests llevados a cabo con el objetivo de mostrar los resultados obtenidos, los cuales corroboren el buen funcionamiento del protocolo en versión 0.
  - *Escenario Real con 5G y Wi-Fi:* realización de una pequeña prueba con equipos reales en la Universidad.
- *Throughput en MPTCP versión 1:* se realizan las mismas pruebas que en la versión 0.
- *Complicaciones Encontradas a lo largo del Proyecto:* se describen las dificultades encontradas en el desarrollo de la parte práctica del proyecto.

- **Capítulo 7. Conclusiones y Líneas Futuras:** este capítulo final consta de 2 partes:
  - **Conclusiones:** se hace un breve comentario acerca de los resultados obtenidos tras las pruebas del capítulo anterior, y se describen las ideas principales que se pueden extraer tras su visionado.
  - **Trabajos Futuros:** se hace un análisis de las posibles actuaciones para la futura mejora del protocolo.

Capítulo 2

Estado del Arte

En el presente capítulo se tratan de exponer algunos de los principales casos de uso del protocolo MPTCP, junto con una serie de proyectos recientes y/o actuales a nivel internacional que también han hecho y hacen uso de él. Por último, se aborda el ámbito más práctico haciendo una breve descripción de las diferentes implementaciones que se han llevado a cabo para este protocolo.

2.1. Casos de Uso

MPTCP es un protocolo que ofrece muchas posibilidades de aplicación. Algunos de los principales casos de uso se detallan en el RFC 8041 [6] y se resumen a continuación:

1. **Distribución de carga entre red celular y Wi-Fi**: este ámbito es donde más es utilizado este protocolo. Generalmente, un dispositivo está conectado a la red Wi-Fi o a la red celular únicamente, y cuando ésta se sobrecarga, el usuario nota una bajada de la velocidad y disponibilidad de los datos. Para evitar esto, se pretende el uso de ambas redes simultáneamente con el fin de distribuir el tráfico y hacer la comunicación más estable y eficiente.
   Un ejemplo de esto lo da Apple para todas las aplicaciones en *Iphone Operating System* (iOS), usando el protocolo como soporte de la aplicación de reconocimiento de voz o Siri [7]. Así, se proporciona distribución de tráfico entre ambas redes mientras los usuarios se mueven físicamente interactuando con Siri.

2. **MPTCP en redes WLAN**: otra aplicación del protocolo ampliamente utilizada y que se puede extraer de la anterior, se refiere al tiempo en el que un dispositivo en movimiento está sin conexión al pasar de obtener la cobertura de un determinado punto de acceso o

Access Point (AP) a otro. Si dicho dispositivo tuviese varias interfaces inalámbricas, conectadas cada una a un AP diferente, el periodo de indisponibilidad podría reducirse e incluso desaparecer. El artículo [8] hace un estudio de esta situación.

3. **MPTCP en un Centro de Procesamiento de Datos (CPD)**: este ámbito de aplicación no es tan común, pero ofrece grandes ventajas. En un CPD se generan redes de gran complejidad debido a la conexión de múltiples dispositivos, tales como servidores, conmutadores (*switches*) y encaminadores (*routers*) con altas prestaciones; además de que entre ellos suele haber conexiones redundantes. Si se instala MPTCP, es posible llevar a cabo una mejor utilización de la red, repartiendo la carga entre los diferentes caminos y mejorando así la tasa de datos, los retardos o la disponibilidad. En el artículo [9], llevado a cabo por una serie de universidades europeas, se reflejan algunos de los beneficios que puede conllevar este uso.

2.2. Proyectos Relevantes

Además de las situaciones que se han mostrado en la sección anterior, en las cuales se usa MPTCP, hay proyectos más grandes y aplicaciones en los que el uso de este protocolo es fundamental, ya que trae consigo una gran cantidad de ventajas. A continuación, se describen brevemente algunos de ellos y su importancia en el mundo de las telecomunicaciones:

- ***Measurement and Arquitecture for a Middleboxed Internet* (MAMI)** [10]: este proyecto surgió en 2016 y se cerró en 2018. Su objetivo era buscar el equilibrio entre la privacidad del usuario final frente a la constante supervisión y la innovación en los protocolos de red frente a la creciente estrictez. Para ello, se buscó la implementación de un *Middlebox Cooperation Protocol* (MCP), protocolo para comunicación entre nodos intermedios de una conexión [11], en una capa de transporte versátil. Es aquí donde entró en juego el uso de MPTCP en esta capa.

  Este proyecto cuenta con su página oficial en GitHub [12], donde se puede encontrar código fuente del software desarrollado en el proyecto, documentación técnica, informes de investigación, presentaciones y otros recursos.

- **5G-MEDIA** [13]: proyecto europeo que tiene como objetivo investigar el posible beneficio que tiene el uso de 5G en las aplicaciones audiovisuales. La importancia de MPTCP se da en la transmisión de vídeo en estas redes, con el objetivo de mejorar la calidad y reducir la latencia, aprovechando las capacidades de la red que haya disponibles.

- ***Multipath Quick UDP Internet Connections* (MPQUIC)** [14]: proyecto/protocolo desarrollado por la Universidad Católica de Louvain en 2017. Surge de *Quick UDP Internet Connections* (QUIC) [15], un protocolo de transporte desarrollado por Google que intenta mejorar el rendimiento y seguridad de las aplicaciones web mediante el uso de *User Datagram Protocol* (UDP) (RFC 768 [16]). QUIC está definido en el RFC 9000 [17] y MPQUIC es una extensión de este protocolo que permite a los dispositivos comunicarse entre sí a través de múltiples redes mediante una sola conexión, por lo tanto, usando funcionalidades de MPTCP. En la figura 2.1 se muestra un ejemplo básico de su objetivo, encaminar los datos por diferentes redes:

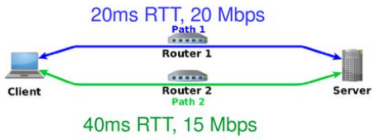


Figura 2.1: Objetivo de MPQUIC [18].

Esta comunidad ha tenido un alcance muy grande en su investigación, hasta el punto de crear una aplicación iOS llamada *Multipath-Tester* [19] cuyo objetivo es evaluar el rendimiento de MPTCP frente a MPQUIC bajo diferentes condiciones. Actualmente, sigue realizando investigaciones en este campo con el fin de mejorar la extensión.

- **MPTCP App** [20]: una empresa tecnológica llamada Tessares [21] y fundada en marzo de 2015, creó una aplicación en el año 2020 llamada MPTCP App para iOS. Esta aplicación permite probar las funciones *Multipath* integradas en iOS, las cuales hacen que el cliente no se vea afectado cuando un dispositivo con este sistema operativo se conecte a una red Wi-Fi y a una red móvil, y se cambie de una a otra buscando la mayor eficiencia. Su interfaz de usuario se observa en la figura 2.2.

Figura 2.2: Interfaz de Usuario de MPTCP App [20].

- ***Cross-layer Cooperation to Boost Multipath TCP Performance in Cloud Networks***[22]: este estudio surgió en 2013 y se publicó en el *Institute of Electrical and Electronics Engineers (IEEE) 2nd International Conference on Cloud Networking (CloudNet)*; a día de hoy sigue en constante desarrollo e investigación. Pretende mejorar el rendimiento y eficiencia de las redes en la nube mediante el uso de MPTCP. Está en desarrollo ya que MPTCP no es del todo eficiente para la nube debido a que crearía una especie de red mallada con enlaces entre numerosas *Internet Protocol* (IP) que provocarían mucha sobrecarga. Actualmente, se trabaja para mejorar la forma en la que se crean los subflujos, con el objetivo de que cree un número adecuado de estos teniendo en cuenta la variedad de rutas que ofrece un protocolo de mapeo de IP en IP, como el *Location/Identifier Separation Protocol* (LISP).

  Así, ha surgido una nueva versión de MPTCP, que es *Augmented Multipath TCP* (A-MPTCP). Su funcionamiento se puede ver en la figura 2.3.

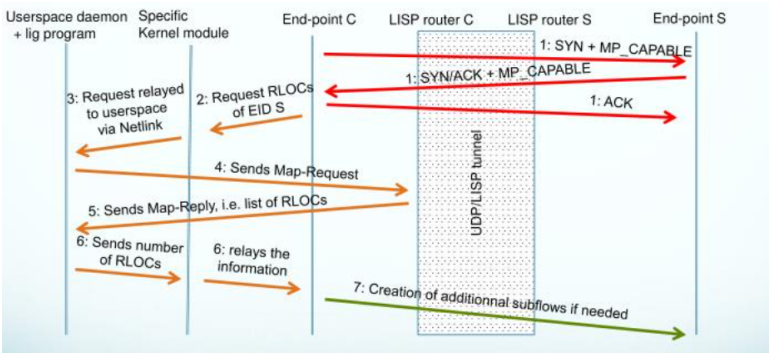


Figura 2.3: MPTCP en el *Cloud* - A-MPTCP [23].

Primero se produce un *three-way handshake* inicial con las opciones MPTCP (esto se explicará en más detalle en el capítulo 3) entre los dos dispositivos. Seguidamente, se produce el mapeo en sí: el extremo iniciador solicita el *Routing Locator* (RLOC) del otro extremo aportando el *Endpoint Identifier* (EID). El RLOC es un identificador que se utiliza para enrutar los paquetes de datos a través de la red, y el EID es otro identificador que se utiliza para reconocer de forma única un dispositivo de extremo específico, independientemente de su ubicación física. Esta solicitud se transmite al *kernel* y este la envía al demonio de espacio de usuario (este demonio se utiliza para realizar tareas en segundo plano que no requieren acceso directo al hardware o al sistema operativo), quien la reenvía al router asociado a este extremo. Este router le devuelve al demonio el RLOC o la lista de RLOC

para llegar al dispositivo que se quiere; en ese momento, el demonio de espacio de usuario lo comunica al dispositivo en sí. Finalmente, cuando el dispositivo que quería iniciar el nuevo camino ya tiene cómo llegar al otro extremo, crea el subflujo entre ambos.

Los resultados muestran que esta nueva versión del protocolo reduce considerablemente el tiempo de transmisión y recepción de los datos, por lo que se vaticina un buen rendimiento en las comunicaciones en la nube o *Cloud* entre usuarios y centros de datos.

- ***Dual-LTE MPTCP with In-Car Voice Assistant***: este es otro proyecto llevado a cabo por un equipo de investigadores de la Universidad de Ciencia y Tecnología de China, y fue publicado en el *IEEE Access Journal en 2021* [24]. En este proyecto se utilizó MPTCP para mejorar la calidad de servicio o *Quality Of Service* (QoS) del asistente de voz de un vehículo aprovechando la conexión de dos redes *Long Term Evolution* (LTE). Para ello, se empleó MPTCP distribuyendo la carga de tráfico entre las dos redes y evitando así la congestión. Esto se probó en diferentes escenarios de red vehicular (teniendo varias aplicaciones en el asistente, o cambiando la velocidad del vehículo) y se comprobó que la QoS mejoraba en cuanto a número de éxitos en las transacciones de voz y bajo tiempo de respuesta, con respecto a escenarios con conexión a una sola red LTE.
  En la figura 2.4 se puede apreciar un escenario de este estilo, con los vehículos conectados entre sí, a la red Wi-Fi y a la red móvil, todo simultáneamente.

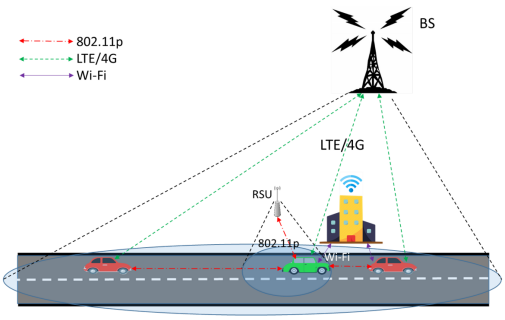


Figura 2.4: MPTCP en Redes Vehiculares [25].

2.2.1. MPTCP en la Industria 4.0

Para terminar de mostrar el alcance que tiene este protocolo en la revolución tecnológica actual y además, con ánimo de exponer el papel que ocupa la Universidad de Granada en la Industria 4.0, se muestra uno de los proyectos más importantes a nivel europeo del que forma parte MPTCP.

Este proyecto se conoce bajo el nombre de *Beyond 5G Multi-Tenant Private Networks Integrating Cellular, Wi-Fi, and Light Fidelity (Li-Fi), Powered by Artificial Intelligence and Intent Based Policy* (5G-CLARITY) [26] y en él participan numerosas empresas, centros de investigación y universidades, entre ellas, la Universidad de Granada.

5G-CLARITY propone un sistema para redes privadas más allá del 5G que integra tecnologías 5G, Wi-Fi y Li-Fi y que está gestionado mediante redes autónomas basadas en Inteligencia Artificial (IA). Está formado por dos pilares básicos: una red de acceso inalámbrico heterogénea que integra tecnología 5G más allá de *Release 16* (R16), Wi-Fi y Li-Fi, y un plano de gestión basado en los principios de las redes definidas por software o *Software Defined Networking* (SDN) y la virtualización de funciones de red o *Network Function Virtualization* (NFV), impulsado por algoritmos de IA.

En la figura 2.5 se presenta la arquitectura del sistema que se propone. Se observa una red privada con un despliegue *multi-Wireless Access Technologies* (multi-WAT) y con una gran variedad de dispositivos de usuario final, estos emplean ingeniería MPTCP para hacer la conexión a las diferentes tecnologías de acceso mediante todas sus interfaces. Dentro de esta red privada se produce la agregación inteligente de las interfaces 5G/Wi-Fi/Li-Fi, dando lugar a servicios *Enhanced Mobile Broadband* (eMBB) y *Ultra-Reliable & Low Latency Communications* (URLLC), y a capacidades de área efectiva superiores al *3rd Generation Partnership Project Release 16* (3GPP R16). Fuera de ésta, hay MNOs con tecnología 5G, un operador que gestiona la infraestructura heterogénea desplegada dentro y múltiples segmentos de infraestructuras y servicios.

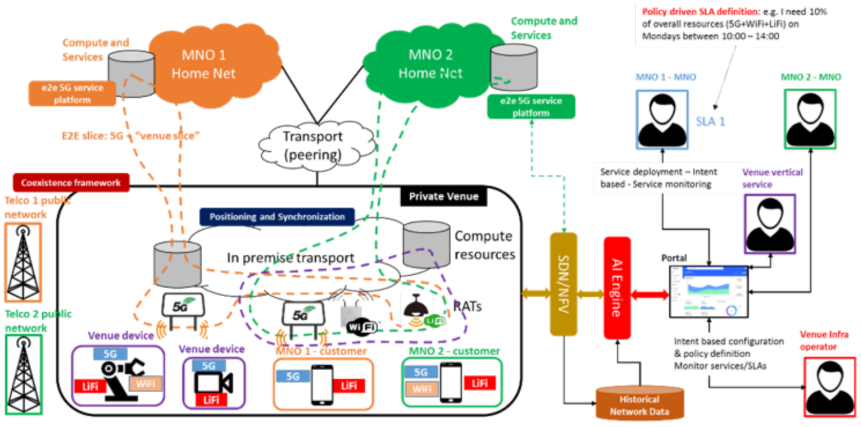


Figura 2.5: Arquitectura del Sistema 5G-CLARITY [26].

Los objetivos que se pretenden lograr con este sistema son: proporcionar servicios de conectividad dentro de la red privada e impulsar la automatiza-

ción de la red, la gestión optimizada de recursos, y la rápida satisfacción de los usuarios.

Para validar este proyecto a un nivel menos complejo antes de pasar a la realidad de la infraestructura mencionada, en la Universidad de Granada se han llevado a cabo una serie de experimentos tanto a nivel virtual (expuestos paso a paso en un repositorio de GitHub [27]), como real [28].

2.3. Implementaciones Disponibles

A continuación, se enumeran las 5 implementaciones independientes más importantes con las que cuenta el protocolo actualmente:

1. ***Kernel* de Linux**: esta implementación tiene dos portales en línea que se pueden consultar si se requiere información. Uno de ellos aborda la implementación de MPTCP versión 0 hasta el *kernel* v5.5 [29] y el otro aborda un proyecto de GitHub en desarrollo bajo el nombre de *Linux MPTCP Upstream Project* [30] que incluye *kernels* modificados desde la v5.6 con MPTCP versión 1 soportado por defecto y una serie de funcionalidades más.

2. ***Citrix Load Balancers***: en la página oficial de Citrix [31], se puede consultar como se habilita y configura el protocolo en esta solución.

3. ***Apple* iOS y *Apple Macintosh Operating System* (MacOS)**: da una breve descripción del uso de MPTCP en sus dispositivos [7].

4. ***Free Berkeley Software Distribution* (FreeBSD)**: *FreeBSD Foundation* ofrece un enlace a los diferentes parches creados para poder utilizar este protocolo en su sistema operativo [32].

5. **Oracle Solaris**: también cuenta con soporte al protocolo a partir de su versión 11.2. Un ejemplo de configuración de MPTCP en sistemas Oracle Solaris se da en el escrito [33], donde se describe cómo configurar MPTCP en redes de alta velocidad que se utilizan para conectar dispositivos de almacenamiento de datos. Estas redes se denominan *Storage Area Network* (SAN).

Se puede observar cómo hay implementaciones tanto para sistemas libres, como para sistemas comerciales. Las 3 primeras interoperan entre sí; además, FreeBSD e iOS y MacOS de *Apple* son de código abierto. La implementación de *Apple* está muy extendida debido a que esta empresa fue la primera en promover y utilizar esta tecnología; sin embargo, la más utilizada es la del *kernel* del Linux, y por ende, es en la que se centra este proyecto.

Capítulo 3

Fundamentos Teóricos

En este capítulo se hace una introducción a los dos protocolos en los que se basa este proyecto, TCP y MPTCP. Para este último se tratan además sus dos versiones (MPTCP versión 0 y versión 1). Por último, se describe con más detalle la implementación que se va a utilizar, el *kernel* de Linux.

3.1. Protocolo TCP

Este protocolo es el pilar fundamental para la realización del proyecto (fiable, eficiente y adaptable a los cambios en la red) ya que MPTCP surge como extensión de él. Por ello, se expondrán sus nociones más básicas con el fin de comprender mejor MPTCP a posteriori.

Está definido en el RFC 793 [34] y es el protocolo de transporte por excelencia usado en TCP/IP (RFC 1180 [35]) junto con UDP. Los segmentos TCP llevan una cabecera de entre 20 Bytes y 60 Bytes (dependiendo de si se utiliza el campo de opciones). A continuación, en la figura 3.1, se muestra la cabecera y sus campos más relevantes:

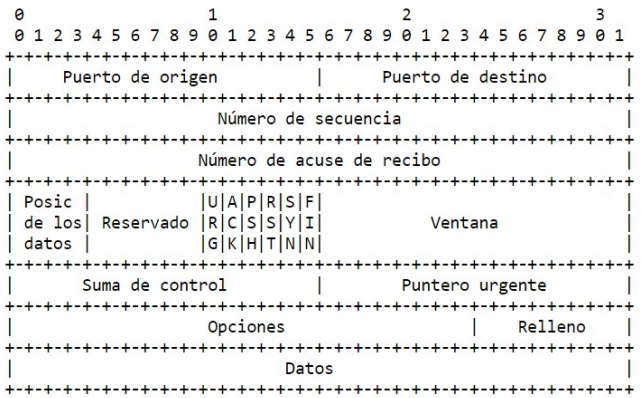


Figura 3.1: Cabecera TCP [34].

- El campo de la cabecera **Número de secuencia** está formado por 32 bits necesarios para identificar los datos que se han transmitido. Cuando se inicia una sesión, se establece a un número aleatorio llamado *Initial Sequence Number* (ISN), el cual se actualiza cada 4 horas y 46 minutos aproximadamente. Es un mecanismo fiable frente a coincidencias, pero es fácil de averiguar.
- El campo de la cabecera **Número de acuse de recibo** (32 bits) se utiliza tanto para indicar al otro extremo de la conexión que ha llegado cierta información, como para señalar el próximo número de secuencia que se espera recibir. También es utilizado en mecanismos de congestión de red.
- Los **bits de control** (6 bits) se emplean para indicar el tipo de mensaje TCP que se está enviando. Hay varios:
  - *Urgent* (URG): indica el uso del puntero urgente.
  - *Acknowledgment* (ACK): indica el uso del campo ACK.
  - *Push* (PSH): indica el envío de datos.
  - *Reset* (RST): se emplea para reiniciar la conexión.
  - *Synchronize* (SYN): se utiliza para sincronizar números de secuencia.
  - *Finalize* (FIN): es utilizado cuando el emisor no va a enviar más datos.
- Otro campo importante es el de **Ventana** (16 bits), utilizado para el control de flujo. Con este campo, el receptor informa al emisor sobre la cantidad de datos que puede recibir sin que su *buffer* (espacio de memoria en el que se almacenan datos de manera temporal) se colapse.
- El campo **Suma de control** (16 bits) es utilizado por el receptor de la información para verificar la integridad de ésta. Realiza una comparación entre la verificación calculada en el emisor y la que él calcula, y observa las coincidencias.
- También se observan otros campos como el de **Puntero Urgente**, **Relleno** y **Opciones**. En este último se encuentran a su vez una serie de campos más, entre ellos ***Type*** y ***Subtype***. Estos campos se explican en la sección 3.2 y se utilizarán para indicar el uso de MPTCP en las comunicaciones.

Los conceptos más relevantes de este protocolo se detallan a continuación:

- Ofrece un servicio punto a punto no multicast, es decir, comunicación entre dos *sockets* (dos conjuntos de IP-puerto).

- Opera en transmisión full-dúplex (los dispositivos TCP transmiten y reciben datos al mismo tiempo).
- Incluye mecanismos de detección y recuperación de errores o *Automatic Repeat Request* (ARQ) con confirmaciones positivas (ACK) acumulativas y tiempos de espera adaptables, mecanismos de control de flujo y también de control de congestión.
- Es un protocolo orientado a conexión, esto es, que hay un proceso de establecimiento de conexión o *handshaking* previo al envío de datos, y un proceso de cierre de conexión.

Los principales procesos que se llevan a cabo en el protocolo son:

- **Intercambio de datos full-dúplex**: una vez que la sesión se ha establecido, ésta se identifica por un *socket* de forma que cada dispositivo pueda tener múltiples sesiones activas sin que se produzca ningún conflicto.
- **Control de congestión**: ajusta la velocidad de tráfico generado a los recursos disponibles en la red, adoptando un procedimiento de prueba y error (la opción de sobredimensionar no siempre mejora el rendimiento de la red). Hay 4 tipos de ajuste [36], y todos ellos hacen referencia al modelado de la tasa de generación de tráfico. En concreto, si se pone atención a la cantidad de bytes permitidos a enviar, se utiliza el control de congestión TCP-Tahoe o TCP-Reno [37], el cual se puede observar en la figura 3.2:

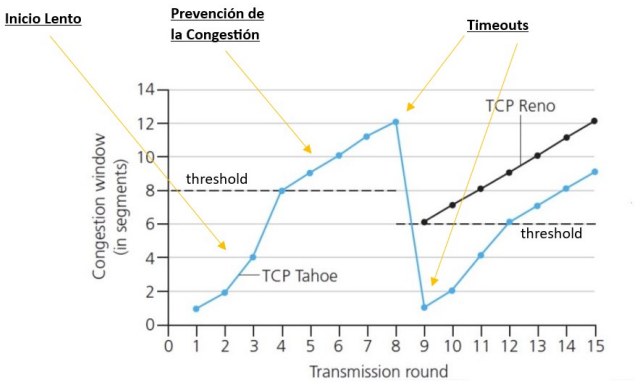


Figura 3.2: TCP-Tahoe vs. TCP-Reno.

En este caso, los bytes permitidos a enviar serán el mínimo entre la Ventana de Congestión (bytes que el emisor puede enviar sin recibir confirmación) y la Ventana del Receptor (bytes que el receptor puede recibir si colapsar su *buffer*). Como se observa, hay 4 momentos clave en este proceso:

- Inicialmente, la Ventana de Congestión vale 1.
- A continuación, como ésta es más pequeña que el Umbral (*threshold*), por cada ACK recibido aumenta en 1; por lo que su crecimiento es exponencial y, por tanto, se denomina *Inicio Lento*.
- Seguidamente, va creciendo hasta superar el Umbral, a partir de ese momento aumenta en 1 cada vez que se reciban todos los ACKs pendientes. Ahora la ventana tiene un crecimiento lineal y se encuentra en *Prevención de la Congestión*.
- Finalmente, la ventana se llena, se produce una expiración de temporizador y se pasa a tener congestión.
  - En caso de Tahoe, la Ventana de Congestión vuelve a valer 1 y el Umbral pasa a ser la mitad de la Ventana de Congestión.
  - En caso de Reno, el Umbral pasa a valer la mitad de la Ventana de Congestión, pero ahora ésta no vale 1, sino el valor del Umbral.
- **Control de flujo**: procedimiento para evitar que el emisor sature al receptor enviando mucha información y/o muy rápido. Para ello, el receptor informa al emisor sobre los bytes que puede transmitirle haciendo uso del campo 'Ventana' de la cabecera TCP y sin esperar respuesta.

  Cuando el receptor libera su *buffer*, avisa al emisor de que puede volver a enviar datos. En caso de que este aviso se pierda, se utiliza el *Timer de persistencia* [38], que en el momento que expira hace que el emisor envíe un byte de prueba para comprobar si el receptor sigue con el *buffer* lleno, o ya ha liberado espacio y puede desbloquearse y enviarle datos.
- **Control de errores**: en caso de que se produzca la pérdida de una confirmación enviada por un extremo de la comunicación, saltará el temporizador o *timeout* y el otro extremo volverá a transmitir la trama. Se basa en escenarios de retransmisión. En caso de errores con las tramas de datos, se 'juega' con el envío de ACK para solucionarlo.
- **Establecimiento de conexión**: el método utilizado es el conocido como *three-way handshake* o apretón de manos, este nombre se debe a que el establecimiento se lleva a cabo en tres pasos o fases. En la figura 3.3 se puede observar dicha secuencia.

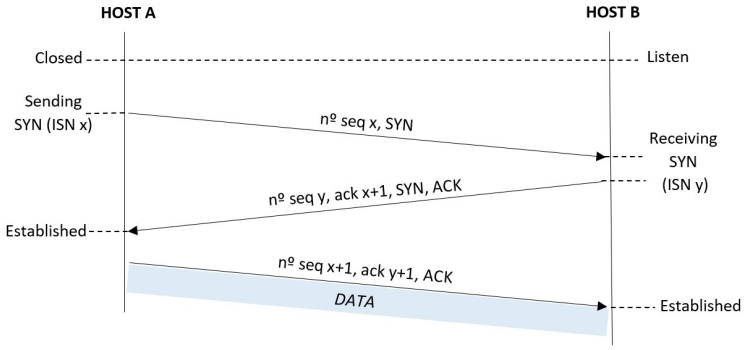


Figura 3.3: Establecimiento de Conexión TCP.

El extremo que desea iniciar la conexión envía un paquete al otro extremo con el bit SYN activado; el receptor del paquete responde mandando un mensaje con los *flags* SYN y ACK indicando que desea continuar con el proceso de establecer la conexión. Finalmente, el primer dispositivo responde confirmando con un ACK y la conexión se establece.

- **Cierre de conexión**: en el momento en el que una aplicación no tenga más datos que transmitir o no desee seguir en la comunicación, se inicia el proceso de cierre de conexión consistente en un *four-way handshake*. De forma similar al proceso anterior, el extremo que quiere cortar la comunicación envía un mensaje FIN/ACK, el otro extremo lo confirma con un ACK y envía también su propio FIN/ACK. Por último, el extremo inicial confirma el cierre con un ACK final y espera un tiempo *MaximunSegmentLifetime* (MSL) de 2 minutos aproximadamente para cortar la conexión. Cuando la otra aplicación recibe la confirmación del mensaje FIN/ACK, cierra inmediatamente la comunicación. En la figura 3.4 se observa esta secuencia:

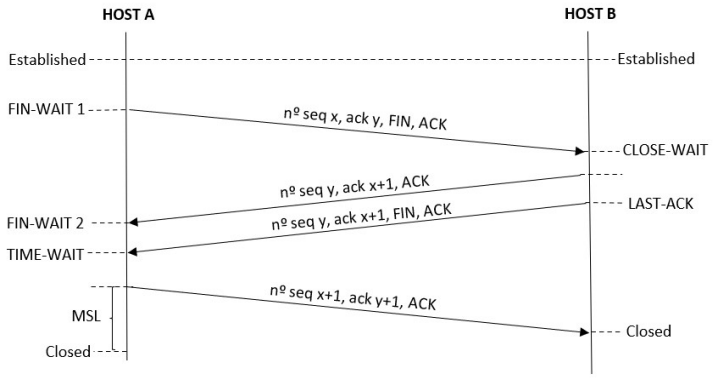


Figura 3.4: Cierre de Conexión TCP.

3.2. Protocolo MPTCP

Multipath TCP es un protocolo de transporte (siguiendo el modelo TCP/IP) que se implementa en las opciones de TCP. En la figura 3.5 se puede observar el lugar que ocupa:

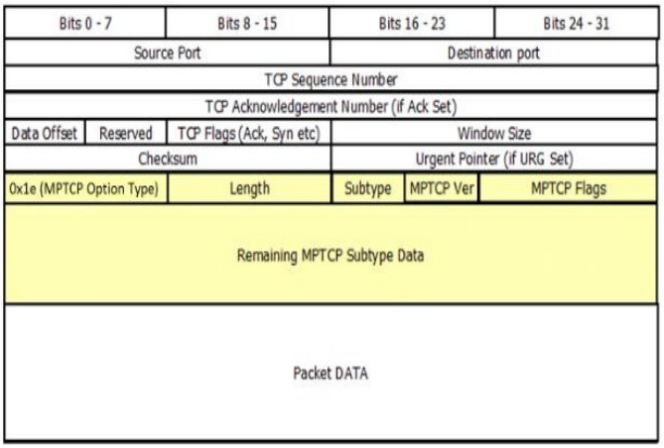


Figura 3.5: MPTCP en el Campo de Opciones de TCP [39].

Toma el valor 30 (0x1e en hexadecimal) en el campo ***Type***, sus opciones van incluidas en el campo ***Subtype*** con distintos valores, y se incluye un campo de ***Length***. Además, hay tres campos más, un campo ***Version*** donde se indica qué versión del protocolo se está utilizando, un campo con ***flags***, y por último un campo con los datos que se necesiten enviar dependiendo del tipo de mensaje MPTCP que se envíe.

Como se mencionó en el capítulo 1, su objetivo es aprovechar todas las interfaces de red disponibles en un dispositivo, estableciendo una única sesión MPTCP que envía y recibe datos empleando las rutas disponibles entre los dispositivos de una comunicación. En la figura 3.6 se puede ver la posición que ocupa en el modelo de capas TCP/IP:

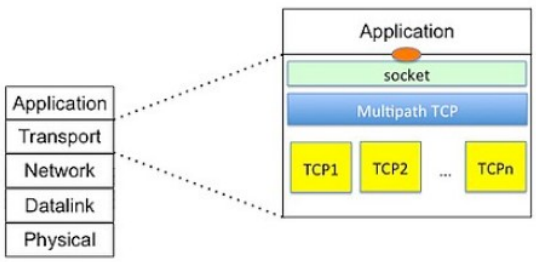


Figura 3.6: MPTCP en el Modelo TCP/IP [40].

3.2.1. Conceptos Fundamentales

Antes de comenzar a exponer el funcionamiento y procesos fundamentales por los que se rige este protocolo, es importante dejar claro una serie de conceptos, con el fin de evitar confusiones. La terminología se describe a continuación:

- Un **subflujo** es un flujo de segmentos TCP entre dos *sockets* IP-puerto, como una conexión TCP normal. Los paquetes que se encaminan por estos subflujos llevan opciones MPTCP que permiten utilizar este protocolo. El conjunto de subflujos entre dos pares es lo que forma un flujo MPTCP.
- Un **flujo** MPTCP es un conjunto de uno o más subflujos a través de los cuales se encaminan los datos. Este conjunto se aprecia como una única conexión entre dos dispositivos.
- Un ***token*** o también denominado ID de la conexión es un identificador único de 32 bits dado por un *host* a una conexión de rutas múltiples.

En la figura 3.7 se puede ver una comunicación MPTCP entre dos dispositivos, esta comunicación está formada por dos subflujos (tratados como dos conexiones TCP normales) los cuales forman un único flujo MPTCP entre ambos extremos.

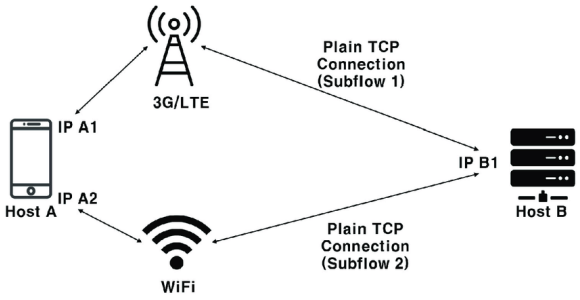


Figura 3.7: Visión General del Protocolo MPTCP [41].

3.2.2. Requisitos de Diseño

Para que MPTCP fuese viable y proporcionase ventajas frente a TCP, en el ejercicio de su desarrollo se establecieron una serie de aspectos que debía cumplir el protocolo una vez implementado. Estos se exponen a continuación:

1. Debía ser compatible con todas las funciones y mecanismos de TCP.
2. Para ofrecer ventajas frente a TCP, debía elegir la mejor o mejores rutas en función de una serie de condiciones.

3. Una sesión MPTCP con un solo subflujo debía tener la misma eficiencia, como mínimo, que una sesión TCP.

4. El protocolo debía ser capaz de reaccionar ante cambios en la topología de red, por ejemplo, manteniendo la sesión activa y mandando información por los demás subflujos a pesar de que uno de ellos se caiga.

5. MPTCP debía tener compatibilidad hacia atrás (*fallback*), es decir, debía poder volver a TCP por sí mismo en caso de comunicarse con dispositivos que no lo soportasen.

6. Tendría que ser transparente para las aplicaciones teniendo la *Application Programming Interface* (API) de *sockets* igual que para TCP. Además, debería proporcionar la misma seguridad que su antecesor.

7. Por último, debía funcionar a través de dispositivos intermedios como *firewalls* (sistema que supervisa y controla el tráfico de red entrante y saliente en función de unas reglas de seguridad ya predefinidas) o *proxies* (intermediario en la comunicación entre dos nodos), entre otros. Por ello, estos debían permitir la opción 30 en las opciones de TCP, que es la que identifica al protocolo.

Este protocolo es considerado como una opción muy válida y fiable ya que en el peor de los casos, se comportará como TCP.

3.2.3. MPTCP Versión 0

Una vez se ha explicado el lugar que ocupa MPTCP dentro de TCP y sus conceptos y requisitos de diseño fundamentales, en la presente sección se exhiben las características principales del protocolo en sus inicios, la versión 0. Como se mencionó en el capítulo 1, esta versión está definida en el RFC 6824 [2]:

Establecimiento de la Conexión

El establecimiento de la conexión MPTCP sigue el mismo patrón que en TCP, como se mostró en la figura 3.3. Sin embargo, hay una diferencia y es que ahora los mensajes SYN, SYN/ACK y ACK van acompañados de la opción MP_CAPABLE activa. Esta opción es utilizada para verificar si el *host* destino es compatible con MPTCP y para permitir a los miembros de una comunicación intercambiar información que ayude a añadir y establecer subflujos adicionales. Como se puede ver en la figura 3.8, su longitud no es fija, depende de la longitud que tengan sus campos.

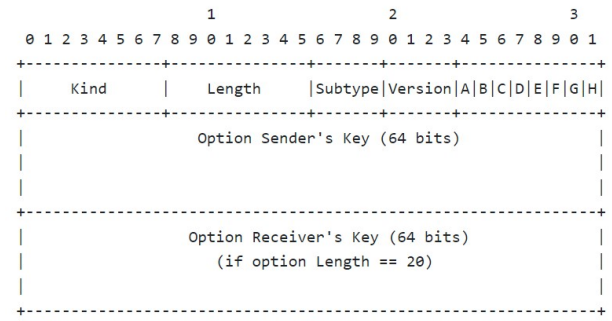


Figura 3.8: Opción MP_CAPABLE en MPTCP Versión 0 [2].

En la figura 3.8 se tienen una serie de campos característicos: está el campo ***Kind***, que contiene el valor 30 al ser MPTCP, un campo de longitud o ***Length***, un campo ***Subtype*** que contiene el valor 0x0 asignado por *Internet Assigned Numbers Authority* (IANA) a esta opción, el campo ***Version*** a 0 y seguidamente varios campos con *flags*:
-**A** : puede estar a 1 si se requiere comprobación.
-**B** : debe estar a 0 para esta implementación. Se pondrá a 1 en implementaciones futuras.
-bits de **C** a **H** : generalmente se utilizan para la negociación de algoritmos criptográficos, pero actualmente solo se emplea el último, el bit H, indicando que se utiliza el *Hash-based Message Authentication Code* (HMAC) [42] *Secure Hash Algorithm 1* (SHA1) [43] en posteriores procesos. Si todas estas *flags* están a 0, la opción MP_CAPABLE se considera inválida y se utiliza TCP normal.

A continuación siguen dos campos que tienen una relevancia mayor, son los campos de **clave**. Estas claves identifican de forma exclusiva a origen y destino de la comunicación y se intercambian en texto plano únicamente en este momento, ya que más adelante en procesos en los que se requiera autenticación, como puede ser el caso de querer añadir un nuevo subflujo, lo que se mandan son *tokens* generados en cada extremo mediante un *hash* SHA1 truncado de cada clave; además, para obtener el *Initial Data Sequence Number* (IDSN), se toman los 64 bits menos significativos de este *hash* sobre la respectiva clave.

Estas claves son únicas, puesto que el *host* debe almacenar la relación entre clave y *token* para identificar la sesión MPTCP sobre la que se está actuando. Además, existe un pequeño riesgo de que dos claves diferentes generen el mismo *token*, por lo que es importante que la implementación compruebe la lista de *tokens* de conexión antes de enviar su clave en el mensaje SYN/ACK. En el caso de que se produzca una colisión de *tokens*, el subflujo que se está intentando establecer no se establecería, pero el resto sí seguiría funcionando

con normalidad.

Así, el proceso de establecimiento de la conexión se rige por el flujo de mensajes mostrado en la figura 3.9:

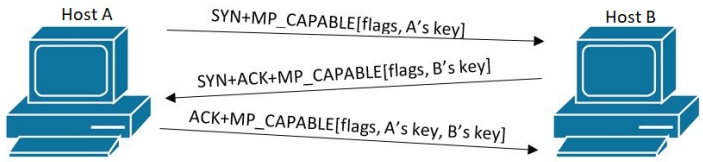


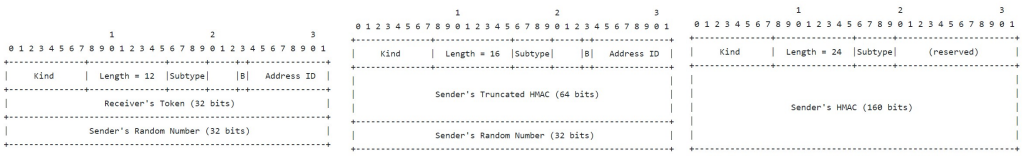
Figura 3.9: Establecimiento de Conexión en MPTCP Versión 0.

El *host* que quiere iniciar la conexión (*host* A) manda el mensaje SYN acompañado de esta opción e incluyendo en ella su clave. Si el *host* B tiene la capacidad de utilizar MPTCP, devolverá el SYN/ACK de la misma forma con la opción MP_CAPABLE y su respectiva clave. Finalmente, el *host* A termina el *three-way handshake* devolviendo a B un mensaje de confirmación con ambas claves. En caso de que el *host* B no tuviese implementado MPTCP, respondería al SYN inicial de A con un SYN/ACK sin la opción MP_CAPABLE y se continuaría con TCP normal.

Gestión de Subflujos

En este apartado se detalla el manejo de subflujos una vez que la sesión MPTCP ya ha sido establecida.

Un *host* conoce sus propias direcciones y también puede conocer las de otro *host* con una serie de mensajes de señalización. Así, es posible empezar un nuevo subflujo sobre un par de direcciones que no se estén usando (esto lo puede hacer cualquiera de los *hosts*, pero generalmente es propio de aquel que inició la conexión).
Para realizar esto se utiliza la opción MP_JOIN, la cual sigue una estructura diferente (figura 3.10) en función del mensaje en el que esté incluida:



(a) MP_JOIN en SYN Inicial. (b) MP_JOIN en SYN/ACK. (c) MP_JOIN en ACK Final.

Figura 3.10: Opción MP_JOIN en MPTCP Versión 0 [2].

El proceso de adición de un nuevo subflujo es parecido a un intercambio TCP de SYN/ACK, pero lleva incorporado esta nueva opción. Los campos más importantes en ella son un ***token*** generado por el emisor tras hacer un *hash* SHA1 de la clave del receptor truncado a los 32 bits más significativos, y usado por este último para identificar a que conexión se está intentando añadir el subflujo, un **número *random*** también conocido como *nonce* y utilizado para evitar ataques de repetición en la autenticación, un código **HMAC** usado para autenticación y generado a partir de las claves que se intercambiaron ambos extremos al inicio de la sesión y de un mensaje *random* (en el SYN/ACK, para el HMAC se utiliza la clave de B seguida de la clave de A y el mensaje es el número *random* de B seguido del de A, el HMAC obtenido se trunca a los 64 bits menos significativos; en el tercer ACK, este HMAC se consigue con la clave de A seguida de la clave de B y el mensaje es el número *random* de A seguido del de B), y por último, un ***Address* ID**. El identificador de dirección reconoce la IP origen de un paquete en una conexión y es usado cuando se producen eliminaciones de direcciones (por ejemplo, cuando hay elementos intermedios) ya que no requiere saber cuál es la dirección origen en el destino. Cada *host* debe almacenar la relación entre el *Address* ID y la dirección asociada a él, tanto para sí mismo como para el otro extremo, así como la relación entre el subflujo asociado y las dos direcciones entre las que se encuentra este.

Como se puede observar en la figura 3.10, esta opción para paquetes SYN incluye un ***flag* B** de 1 bit que se pone a 0 si el nuevo subflujo va a ser un subflujo normal que forme parte de la sesión, o a 1, si va a servir como ruta de reserva en caso de fallo de otros trayectos. Además, para todos los paquetes la opción lleva los campos ***Kind*** (con valor 30) y ***Subtype*** (con valor 0x1).

El intercambio de mensajes para la adición de un nuevo subflujo sigue el esquema mostrado en la figura 3.11. Primero se produce el establecimiento de sesión, como se comentó en la sección anterior. A continuación, el emisor (*host* A) manda un mensaje SYN acompañado de la opción MP_JOIN que incluye el *token* de B, el número *random* generado por él mismo, el *Address* ID y el *flag* B. El *host* B responde con un SYN/ACK más la opción MP_JOIN incluyendo su código HMAC, su número *random*, su *Address* ID y el flag B. Finalmente, emisor responde con un ACK junto con MP_JOIN incluyendo su HMAC, y receptor termina confirmando dicho mensaje.

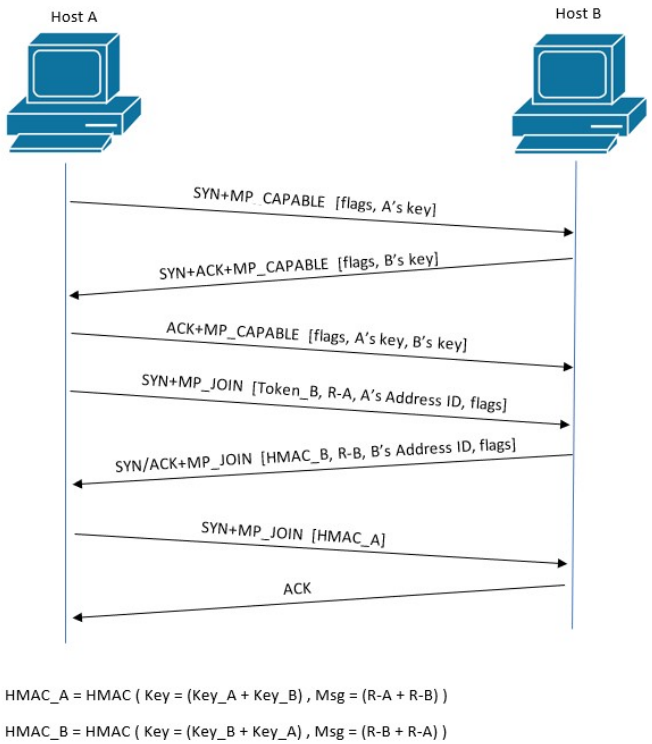


Figura 3.11: Adición de un Nuevo Subflujo en MPTCP Versión 0.

Se pueden dar casos de fallo como:

- Si el *token* recibido en el *host* B es desconocido o ya no se permiten nuevos subflujos, el receptor debe responder con un TCP RST (*reset* de TCP) para dicho subflujo.
- Si el *host* B acepta el *token*, pero el HMAC que manda al *host* A no coincide con el que este espera, el *host* A cierra el subflujo con un TCP RST de nuevo.
- Si el *host* B no recibe el HMAC esperado o no va a incluida la opción MP_JOIN en el tercer mensaje del *handshake*, cierra el subflujo con un TCP RST.
- Si *host* B devuelve el SYN/ACK sin la opción MP_JOIN, *host* A cierra el subflujo con un RST.

Si se verifica que los HMAC son correctos, entonces ambos *hosts* se han autenticado mutuamente y han acordado a qué conexión formará parte el nuevo subflujo.

- Anuncio y Eliminación de Direcciones

Otro concepto a mencionar que se considera importante es el anuncio y eliminación de direcciones en los *hosts*.

A lo largo de una conexión MPTCP, el conjunto de direcciones IP asociadas a un *host* puede cambiar. Dependiendo de cuál sea el motivo, se puede hacer uso de dos métodos diferentes:

1. Si el motivo es la creación de un nuevo subflujo en una dirección IP que no estaba siendo utilizada, el *host* en cuestión manda un MP_JOIN con dirección origen la adicional y dirección destino la IP del otro extremo que se conoció al inicio de la sesión. Esto se presentó en la figura 3.11.

2. Si el motivo es por el anuncio de que hay direcciones disponibles, se puede hacer uso de la opción ADD_ADDR. Esta opción se envía en ACKs (en la figura 3.12 se anuncia la dirección IP_{A2}):

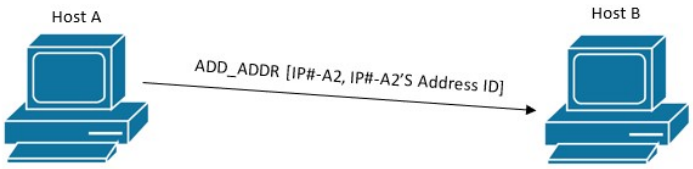


Figura 3.12: Anuncio de una Nueva Dirección en MPTCP Versión 0.

Puede ser utilizada en cualquier momento de la sesión y puede contener varias direcciones siempre y cuando se deje espacio para las demás opciones TCP. El formato que sigue se puede ver en la figura 3.13:

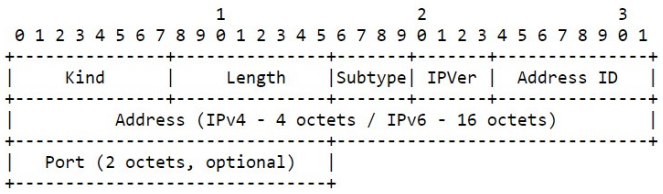


Figura 3.13: Opción ADD_ADDR en MPTCP Versión 0 [2].

Entre los campos relevantes están el campo ***Subtype***, ahora a valor 0x3, ***IPVer*** indicando si se utilizan direcciones IPv4 o IPv6, y el ***Address* ID** como el mencionado en el apartado 3.2.3. Este último se emplea para evitar que se configuren subflujos duplicados en la misma ruta cuando se envían MP_JOIN y esta opción de forma simultánea.

Lo mismo ocurre cuando una dirección deja de ser válida o ya no debe usarse. Primero se debe enviar un mensaje *Keepalive* de TCP en la ruta, para ver si se recibe respuesta. En caso de recibirla, no se procede a su eliminación, pero

sí se puede indicar que esa ruta ya no se usa mediante la opción MP_PRIO, entre otros. Si, por el contrario, no se recibe respuesta, se hace uso de la opción REMOVE_ADDR para eliminarla (ahora el campo ***Subtype*** tendrá valor 0x4). La estructura de esta opción aparece en la figura 3.14.

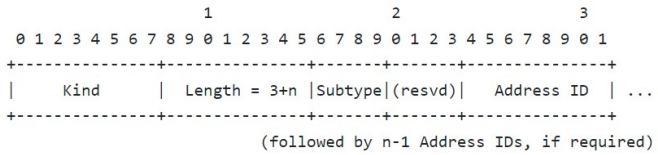


Figura 3.14: Opción REMOVE_ADDR en MPTCP Versión 0 [2].

Cuando se manda esta opción, ambos *hosts* envían RST en los subflujos afectados para cerrarlos. Los mensajes intercambiados se muestran en la figura 3.15:

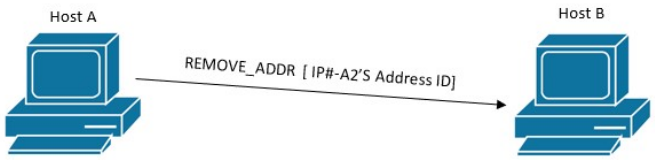


Figura 3.15: Eliminación de una Nueva Dirección en MPTCP Versión 0.

Transmisión de Datos

Una vez se ha establecido la sesión y en caso de que se hayan querido añadir subflujos posteriormente, estén añadidos, es necesario realizar la entrega de datos. Para ello se utiliza la opción *Data Sequence Signal* (DSS) con la estructura de la figura 3.16:

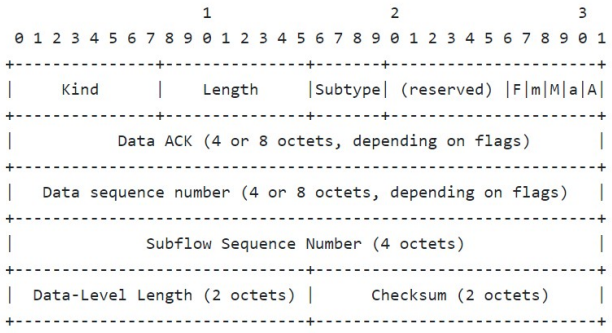


Figura 3.16: Opción DSS en MPTCP Versión 0 [2].

A nivel de aplicación, a la hora de transmitir información solo se ve una única entrada y salida, pero en realidad el protocolo divide esta información entre los diferentes subflujos. Para poder realizar esto de forma fiable y ordenada, se numeran los segmentos añadiendo un nuevo nivel de números de secuencia mediante la opción DSS presentada en la figura 3.16.

En ella se encuentran los campos ***Subtype*** y ***Data* ACK**, el primero a valor 0x2 y el segundo empleado para confirmar la correcta recepción de los datos a nivel de flujo (es análogo al ACK acumulativo de TCP).
También se observa el campo ***Data Sequence Number* (DSN)** de 64 bits, empleado como número de secuencia de datos que identifica a cada segmento dentro del flujo MPTCP, este campo permite al receptor ordenar los paquetes que va recibiendo de los diferentes subflujos. Su valor inicial se establece en el *three-way handshake* del primer subflujo como los 64 bits menos significativos del código *hash* de la clave del emisor, y se va incrementando por cada transmisión.

Los últimos campos son el de ***Subflow Sequence Number* (SSN)**, usado por los subflujos para garantizar el envío ordenado y controlar las retransmisiones, el campo ***Data-Level Lenght***, que es un campo para indicar la longitud de los datos de nivel, y el campo ***Checksum***. Esta suma de comprobación solo está presente si se ha negociado en la opción MP_CAPABLE. En caso de que aparezca, pero no se haya negociado, se ignora; si se ha negociado, pero no aparece, el receptor cierra el subflujo con un RST y lo considera 'roto'.

Además de los campos mencionados, se pueden ver una serie de *flags* que puestos a 1 tienen diversos significados: **A** (hay *Data* ACK), **M** (hay DSN, SSN, *Data-Level Length* y *Checksum*), **a** (*Data* ACK es de 8 Bytes, no de 4 como suele ser) y **m** (DSN es de 8 Bytes, no de 4 Bytes como acostumbra). Los *flags* **a** y **m** solo están presentes cuando **A** y **M** estén activos. Por último se tiene **F** (es como mandar un DATA_FIN, indica que no hay más datos que enviar).

A la hora de transmitir los datos, se tiene la estructura presentada en la figura 3.17. Se observa el *Sequence Number* (SN) y ACK, usados por la conexión para identificar los paquetes que están transcurriendo por ella. Además, en el campo de opciones TCP dentro de la opción DSS, existe una jerarquía de números de secuencia: la más interna, que incluye el SSN y ACK de TCP a nivel de subflujo, y la más externa, con DSN y *Data* ACK a nivel de flujo.

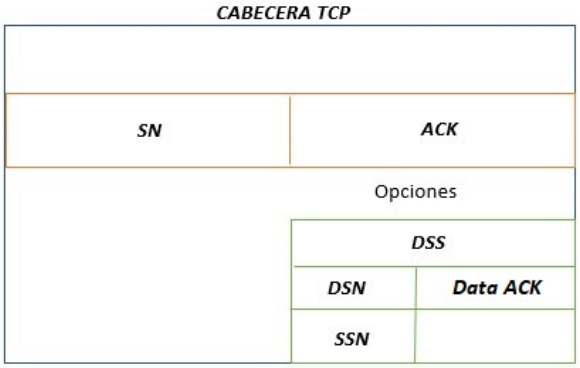


Figura 3.17: Niveles de Jerarquía en la Transmisión de Datos en MPTCP Versión 0.

- Mapeo de Datos

A la hora de recibir el flujo de datos, este se puede reemsamblar utilizando los campos que definen la asignación del *Subflow Sequence Number* (SSN) al *Data Sequence Number* (DSN) en la opción DSS, lo que viene a ser un mapeo del espacio de subflujo al espacio de datos. Este mapeo se lleva a cabo en términos de números de secuencia iniciales para el subflujo y los datos y en una longitud de bytes para los que la asignación es válida. Con ello se busca compatibilidad cuando la segmentación de TCP/IP se lleva a cabo por separado de la implementación que genera el flujo de datos (i.e: los datos se generan en *middleboxes* como *firewalls*, entre otros).

Un extremo transmisor de datos no debe cambiar el mapeo, ya que el número de secuencia de un subflujo está vinculado al número de secuencia de datos después de que se haya producido este mapeo; aun así, diferentes subflujos pueden mapear el mismo número de secuencia de datos si se requiere retransmisión. De esta forma, los mismos datos van a poder retransmitirse por diferentes subflujos en caso de que se busque máxima velocidad o rendimiento. Además, no es necesario realizar un mapeo de secuencia de datos en cada paquete MPTCP siempre y cuando el espacio de secuencia de subflujo en ese paquete se cubra por un mapeo conocido en el receptor; esto reduce gastos en los casos en los que el mapeo ya se conoce antes.

Esta asignación contiene una suma de verificación de datos (*Checksum*) en caso de que haya sido negociada en el intercambio MP_CAPABLE inicial. Se utiliza para comprobar si la carga útil ha sido modificada por algún *middlebox* que no reconoce MPTCP, o si la longitud de los datos en el subflujo ha cambiado, significando que el mapeo de secuencia de datos no es

válido. Al fallar, provoca un error en el subflujo y una vuelta a TCP normal. La estructura de esta suma de verificación es igual que la de TCP, y opera junto con un pseudo encabezado con el esqueleto de la figura 3.18:

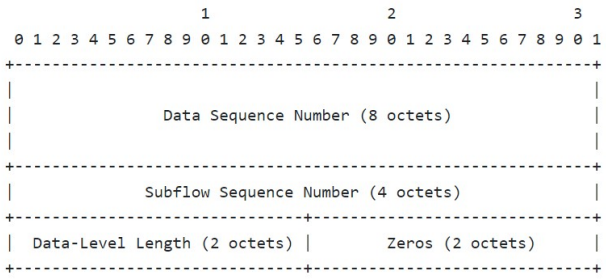


Figura 3.18: Pseudo encabezado para *Checksum* en MPTCP Versión 0 [2].

Como se puede observar en la figura 3.18, es muy parecida a la opción DSS presentada en la figura 3.16. El DSN inicial de cada *host*, al igual que en TCP, no comienza en 0, sino en un número aleatorio denominado IDSN y formado por los 64 bits menos significativos del *hash* SHA1 de su clave. Esto ayuda al receptor a determinar cuál es el IDSN esperado y saber si falta algún paquete de nivel de conexión inicial. En versiones posteriores, este IDSN se forma usando el *hash Secure Hash Algorithm 256* (SHA256) (definido en el RFC 6234 [44]).

Fallback o Retirada

Uno de los requisitos de diseño de MPTCP comentados en la sección 3.2.2 es que en caso de que no se soporte el protocolo, este pueda retroceder y actuar como TCP. Pues bien, hay tres casos que son los más comunes:

1. El primero de ellos ocurre cuando hay una pérdida de opciones TCP y una modificación de los datos de carga útil. Si los dispositivos finales o la ruta en sí no son compatibles con MPTCP, las opciones TCP no llegarán a través de los SYN y el subflujo se cerrará.

2. Otro caso es cuando las opciones MPTCP son admitidas en los mensajes SYN, pero no en los paquetes de datos normales.
   Si esto ocurre en el primer subflujo, el emisor debe incluir la opción DSS explicada en la sección 3.2.3 con asignación de secuencia de datos en cada segmento, hasta que reciba una opción DSS que contenga un ACK de datos. Al recibirla, el emisor puede verificar que el DSS pasa en ambas direcciones. Sin embargo, si el ACK recibido no contiene esta opción incluyendo el ACK de datos, sabrá que la ruta no es compatible con MPTCP, indicará mapeo infinito (mapear los datos de nivel de subflujo a los datos de nivel de conexión para el resto de la conexión) mandando un mensaje DSS con el campo *Data-Level Length* y la

suma de verificación a 0, y volverá a TCP normal enviando los datos de nuevo por el único subflujo que hay sin incluir opciones MPTCP. Si, por el contrario, las opciones MPTCP no son admitidas en los paquetes de datos transmitidos por un subflujo creado posteriormente mediante la opción MP_JOIN, el *host* cerrará el subflujo con un *reset*.

3. Por último, como se explica en la sección anterior y se reitera en el apartado 3.2.3, hay otra situación en la que también se recurre a esta retirada, y es cuando la suma de comprobación con la que están protegidos los datos falla. En dicho caso, también será necesario volver a TCP.

Cierre de Conexión

El proceso por el que se termina una conexión MPTCP es similar al de TCP mostrado en la figura 3.4, pero con una serie de particularidades que se describen a continuación.

Para cerrar la conexión se produce un intercambio de mensajes DATA_FIN en todos los subflujos. Esta señal se indica con el *flag* F dentro de la opción DSS (figura 3.16). Los DATA_FINs de ambos *hosts* son reconocidos por DATA_ACKs cuando se terminan de recibir los datos que queden pendientes. Si alguno de los subflujos tiene datos en cola y no ha devuelto el DATA_ACK, debe cerrarse con un RST, puesto que dichos datos ya se habrán enviado por otros subflujos con el fin de recibir la confirmación del DATA_FIN. Una vez se ha producido esto y para dar por cerrada la conexión, en ambos extremos se deben intercambiar mensajes FIN en cada uno de los subflujos al igual que ocurría en TCP.

En caso de que solo se quiera cerrar un subflujo determinado, se utiliza el mismo mecanismo que TCP (intercambio de FIN). Además, si todos los subflujos se han cerrado con un intercambio FIN, pero no se ha recibido ni reconocido DATA_FIN, la conexión MPTCP se trata como cerrada después de un tiempo de espera. Esto permite escenarios *break-before-make* (véase la sección 3.3.1), donde todos los subflujos se han cerrado y se intenta establecer uno nuevo para no perder la conexión MPTCP.

Otros procesos

Además de los procesos que se han ido detallando en este capítulo, existen otros que también son utilizados con frecuencia, como son:

- **Cambio en la Prioridad de la Ruta (Política de Subflujo)**: se utiliza la opción MP_PRIO. Esta opción cambia la prioridad de los subflujos, si se desea que sirvan de reserva o sean normales.

Generalmente, en las aplicaciones se suele buscar el máximo rendimiento, por lo que se utilizan todas las rutas que hay disponibles para la transferencia de los datos (usando control de congestión); aunque hay veces en las que se prefiere dejar alguna ruta de respaldo por si hay fallos en las demás. Para todo ello, es necesario conocer el coste asociado a cada uno de los enlaces, sin embargo, esto es difícil y lo deseable es que el receptor (que suele ser la parte multiproveedor que paga por el ancho de banda entrante medido) pueda señalar sus propias preferencias de ruta.

La opción MP_JOIN (véase la sección 3.2.3) contiene el bit B, que como ya se explicó anteriormente, permite que un *host* indique a su par que una determinada ruta se va a usar como respaldo (B=1) en caso de fallo de otros subflujos (estos tendrán B=0). Además de esto, si en plena conexión se desea cambiar la prioridad de un subflujo, se puede usar el indicador **B** en la opción MP_PRIO mostrada en la figura 3.19:

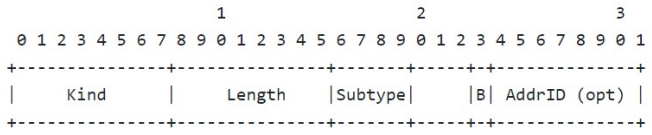


Figura 3.19: Opción MP_PRIO en MPTCP Versión 0 [2].

Además del campo B, hay otros dos campos interesantes en la figura 3.19. El campo ***Subtype*** que tiene valor 0x5, y el campo de ***Address* ID**. Este campo se emplea cuando se quiere que la configuración dada a B se aplique a todos los subflujos que usen la dirección identificada con dicho ID. Este campo depende del campo ***Length***, si *Length*=4, entonces se puede usar, si *Length*=3 no se aplica (la importancia del *Address* ID reside en la posibilidad de establecer como rutas de respaldo aquellos subflujos que usen una dirección que no está disponible).

- **Gestión de fallos**: para indicar un fallo se emplea la opción MP_FAIL. Esta se suele utilizar cuando la suma de comprobación no es válida y no se puede recuperar el subflujo. Ocurre generalmente porque al haber numerosos subflujos, los datos en curso en uno de ellos probablemente incluyen datos que no forman parte del flujo de nivel de conexión y no es posible determinar qué ajustes se han realizado. En ese momento, el subflujo se cierra con un RST que incluye la opción MP_FAIL con la estructura de la figura 3.20:

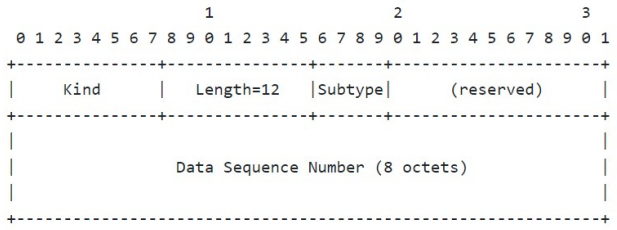


Figura 3.20: Opción MP_FAIL en MPTCP Versión 0 [2].

En caso de que el fallo se produzca en el único subflujo que hay, el *host* detector (receptor) devolverá un ACK de nivel de subflujo junto con la opción MP_FAIL, incluyendo el **DSN** del inicio del segmento en el que se encuentra la suma de verificación fallida. En el otro extremo (emisor), si todos los datos que no se han reconocido son contiguos (esto suele ser así cuando solo hay un subflujo), el *host* esperará ACK de subflujo para borrar su *buffer* de envío, e indicará mapeo infinito. En ese momento, el receptor deberá dejar de generar ACK de datos después de recibir un mapeo infinito.

En caso de que los datos en el subflujo no sean contiguos, ya sea porque contiene datos de otro que se acaba de cerrar, se cierra el subflujo con un RST con MP_FAIL, y se ignoran todos los datos que siguen al número de secuencia de datos especificado.

Además del campo DSN, otro campo importante que se ve modificado en esta opción con respecto a las demás es el campo ***Subtype***, su valor ahora es de 0x6.

- ***Fast Close* o Cierre Rápido** : en TCP, el *reset* o RST cierra la conexión completa, por el contrario, en MPTCP el *reset* funciona a nivel de subflujo, solo termina el subflujo sobre el que se aplica. Es por esto que para terminar de forma abrupta el flujo MPTCP (por ejemplo, cuando el sistema se está quedando sin recursos), se emplea la opción MP_FASTCLOSE mostrada en la figura 3.21 en un ACK.

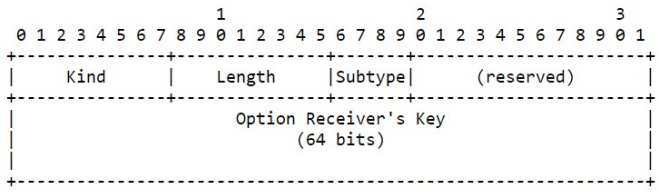


Figura 3.21: Opción MP_FASTCLOSE en MPTCP Versión 0 [2].

Los pasos del proceso se describen a continuación y en la figura 3.22:

1. *Host* A envía un ACK que contiene la opción MP_FASTCLOSE en un subflujo (esta opción contiene la **clave** del *host* B y el valor 0x7 en el campo ***Subtype***). En los demás subflujos, el *host* A envía un TCP RST normal para cerrarlos (pasa al estado FASTCLOSE_WAIT).
2. En el *host* B, al recibir el MP_FASTCLOSE con la clave válida, envía en el mismo subflujo un TCP RST y destruye todos los demás (pasa al estado CERRADO y puede cerrar toda la conexión).
3. Cuando el *host* A recibe el TCP RST en este subflujo, puede cerrarlo y derribar toda la conexión (pasa del estado FASTCLOSE_WAIT al estado CERRADO).

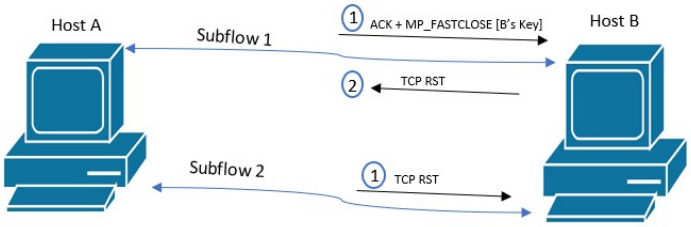


Figura 3.22: Cierre Rápido en MPTCP Versión 0.

Si en el paso 3 el *host* A recibiese un MP_FASTCLOSE en lugar de un TCP RST, debería responder con un TCP RST y cortar la conexión (en este caso, ambos *hosts* estarían intentando el cierre rápido simultáneamente). Si por otro lado, no recibiese nada en respuesta a su MP_FASTCLOSE inicial, después de un tiempo de espera retransmitiría el MP_FASTCLOSE (normalmente no se hacen más de 3 retransmisiones).

- **Control de Flujo** : al igual que en TCP, MPTCP también utiliza una ventana de recepción única compartida entre los subflujos, con la idea de permitir que cualquier subflujo envíe datos siempre y cuando el receptor esté dispuesto a aceptarlos. No se utiliza una ventana por subflujo ya que si se hiciese así, se podrían acabar paralizando algunos subflujos mientras que otros no agotarían su ventana.
  El emisor debe guardar localmente los anuncios de la ventana del receptor, si los tamaños de ésta son diferentes para cada subflujo, este toma el mayor de ellos. Por su parte, el *buffer* de envío debe ser, como mínimo, tan grande como el *buffer* de recepción.

- **Control de Retransmisiones** : en esta especificación no hay un mecanismo concreto que gestione retransmisiones y garantice fiabilidad.

Un concepto importante que sí es conveniente mencionar es que si algunos datos deben retransmitirse varias veces en un subflujo, existe el riesgo de bloquear la ventana de envío. En este caso, el emisor puede decidir terminar el subflujo que se está comportando mal enviando un RST.
También se destaca que los remitentes mantienen los datos que no han sido confirmados en su *buffer* de envío preparados para ser retransmitidos, hasta que se cumple un *timeout*. Estos pueden ser reenviados con el mismo número de secuencia de datos en diferentes subflujos, al hacer esto, el *host* receptor ignorará las retransmisiones adicionales. Esto asegura recepción, aunque no es un proceso ampliamente óptimo.

- **Control de Congestión** : hay varios algoritmos de control de congestión que son utilizados por MPTCP:
  - ***Linked Increase Algorithm* (LIA)**: se rige por el RFC 6356 [45] y su objetivo es mantener un rendimiento al menos tan bueno como el TCP único en el mejor trayecto, no tomar más capacidad que un flujo único TCP, y que los subflujos dependan de la congestión de sus trayectos.
  - ***Oportunistic Linked Increase Algorithm* (OLIA)**: presenta algunas ventajas frente a LIA y al igual que él, se puede usar de forma segura como mecanismo de control de congestión por defecto. Se basa en TCP Reno y actualmente hay un draft publicado en 2014 [46] en el que se explica.
  - ***Balanced Linked Increase Algorithm* (BALIA)**: se basa en TCP Reno y cuando solo hay un subflujo, es idéntico a él. Un ejemplo de las ventajas que ofrece este mecanismo se encuentra en el artículo [47], donde se analiza el uso de MPTCP en *streamings* de vídeo usando diferentes mecanismos de congestión.
  - ***Delay based CC scheme tuned for MPTCP* (wVegas)**: estima el retardo en cola de cada ruta y adapta los pesos de cada subflujo en función de dichos retardos. Además, tiene en cuenta la velocidad de envío y el retardo en cola. Es el mecanismo que más evita la congestión cuando hay cambios de red. Algunas simulaciones realizadas con este esquema de congestión, las cuales prueban sus características, se encuentran en el documento [48].
- ***Path Management* o Gestión de Rutas** : al haber múltiples rutas entre los dispositivos de una red que utiliza MPTCP, es importante saber gestionar de manera eficiente estas rutas y el tráfico que pasa por ellas. En las redes de comunicaciones actuales, cada *Internet Service Provider* (ISP) intenta llevar a cabo ingeniería de tráfico para optimizar la utilización de los recursos dentro de sus redes, pues bien,

se debe tener cuidado con aquellas rutas que son similares y que han sido configuradas por MPTCP, ya que algún ISP podría modificarlas.

Para asegurar la configuración exitosa de subflujos adicionales por parte del protocolo, se emplea la función *path management*, encargada de manejar las rutas que aparecen y desaparecen durante la vida de una conexión. En Linux (véase sección 3.3.1), esta funcionalidad se implementa mediante un módulo o componente del *kernel* denominado *path manager*, quien gestiona la selección de rutas siguiendo una serie de algoritmos (estos algoritmos se describen en el capítulo 5).

- ***Packet Scheduling* o Programación de Paquetes** : este algoritmo es el encargado de fraccionar el flujo de bytes que se recibe de la aplicación en distintos segmentos, y decidir por qué subflujo serán transmitidos mediante una estrategia de programación que dependerá de la información que tenga sobre las rutas disponibles (coste, latencia, rendimiento). Además, se encarga de hacer la reordenación de los paquetes que lleguen de los diferentes subflujos a nivel de conexión, haciendo uso del mapeo de datos. Al igual que el algoritmo anterior, en la implementación en Linux este mecanismo se lleva a cabo mediante un *packet scheduler*, que como se comenta en la sección 3.3.1, puede ser de diversos tipos y estar implementado de forma diferente dependiendo de la versión de MPTCP utilizada (los diferentes *schedulers* se comentan en detalle en el capítulo 5).

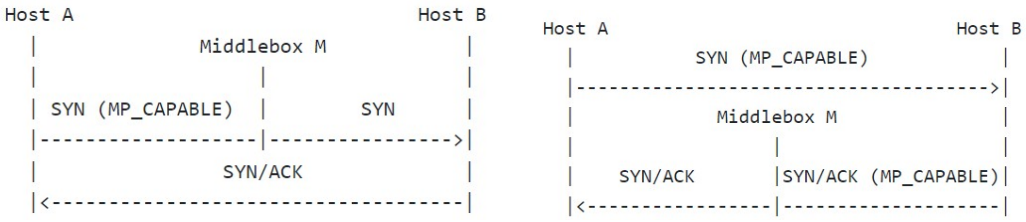
  Este algoritmo, junto con el *path management*, se explican en más profundidad en el RFC 6182 [49]. Además, estos dos, junto con el proceso de control de congestión, funcionan en armonía. El *path management* se encarga del descubrimiento de los múltiples caminos entre *hosts*, una vez establecidos, el *packet scheduling* recibe el flujo de datos por parte de la aplicación, lo segmenta y envía los paquetes por el/los subflujos correspondientes haciendo uso del algoritmo de control de congestión seleccionado (este se emplea para programar qué segmentos deben enviarse, por qué subflujo y a qué velocidad en cada uno). A su salida, el programador de paquetes hace la reordenación a nivel de conexión y manda los datos a la aplicación.

- **Interacciones con *Middleboxes*** : como se puede esperar, los *middleboxes* o Cajas Intermedias son dispositivos que se sitúan en medio de una comunicación entre dos extremos, estos pueden ser un *Network Address Translation* (NAT), *firewalls* o *proxies*, entre otros. Los problemas de estos dispositivos es que no reenvían paquetes con campos desconocidos y pueden reducir el tamaño de la ventana de recepción a nivel de subflujo y segmentar paquetes.

Como MPTCP está diseñado para el Internet actual, contempla tener que lidiar con estos dispositivos, y por ello, emplea campos conocidos de TCP (campo *Type* que se pone a valor 30 y campo *Subtype* para los diferentes tipos de mensajes) con el fin de evitar que, por ser campos desconocidos, los mensajes de este protocolo no sean reenviados por los *middleboxes*. La mayoría de ellos no causan problemas, pero hay otros que sí, en lo referente a haber varios caminos, la presencia del *flag* SYN, la eliminación y copia de opciones de unos paquetes a otros debido a la segmentación que puedan hacer, y la reducción de la ventana de recepción a nivel de subflujo haciendo que aunque MPTCP utilice la ventana máxima a nivel de datos, como se mencionó en el apartado 3.2.3, pueda tener que obedecer a la específica del subflujo.

Casos de Fallo con *Middleboxes*

1. Si un paquete MPTCP SYN con la opción MP_CAPABLE se descarta, se recurre a TCP regular. Esto se observa en la figura 3.23.



(a) Eliminación en Ruta de Salida. (b) Eliminación en Ruta de Retorno.

Figura 3.23: *Middleboxes* Eliminando Opción MP_CAPABLE en MPTCP Versión 0 [2].

2. Si la opción MP_CAPABLE es aceptada, pero la que se descarta es MP_JOIN, la ruta no se utilizará. Al receptor le llegará un SYN normal, a lo que responderá con un SYN/ACK o con un RST. Cuando el *host* que ha querido añadir un nuevo subflujo al flujo MPTCP reciba un SYN/ACK sin la opción MP_JOIN y el *token* correspondiente (como se explicó en la sección 3.2.3), enviará un RST. El establecimiento del subflujo falla, pero la conexión MPTCP sigue, a diferencia de MP_CAPABLE.
3. El comportamiento de MPTCP cuando sus opciones son eliminadas en los paquetes de datos se explica en la sección 3.2.3. Si se pierden algunas asignaciones de secuencias de datos, la conexión continua siempre y cuando haya asignaciones en el nivel de subflujo; si se pierden asignaciones a nivel de subflujo (hay datos sin

mapear), el subflujo afectado se cierra. Además, algunos ACK de datos se pueden perder, disminuyendo el rendimiento a medida que aumente la cantidad de opciones eliminadas.

Algunos de los *middleboxes* más comunes, junto con los problemas que provocan y la solución que da MPTCP, se describen a continuación:

- **NAT** : su comportamiento se define en el RFC 3022 [50]. Estos dispositivos cambian la dirección y puerto de origen de los paquetes, por lo que un *host* no puede conocer su dirección pública para señalización MPTCP. Contra esto, el protocolo permite la adición de direcciones con la opción MP_JOIN, garantizando que los intentos de conexión a una IP privada solo configurarán subflujos en los *hosts* correctos. Para añadir y eliminar direcciones se usa el ID de dirección (en ADD_ADDR y REMOVE_ADDR), evitando tener que conocer la IP origen.
- ***Proxies* de mejora de rendimiento o *Performance Enhancing Proxy* (PEP)** : MPTCP utiliza señales de control de congestión precisas por parte del *host* final, y aquellos PEP que no sigan MPTCP no podrán llevarlas a cabo. Estos *proxies* se definen en el RFC 3135 [51] y para luchar contra esto, este protocolo permite volver a TCP sin problema.
- **Normalizadores de tráfico** : estos dispositivos a veces no permiten huecos en los números de secuencia y almacenan paquetes en caché para posteriormente retransmitirlos. Más información sobre normalización de tráfico se tiene en el escrito [52]. MPTCP no permite la retransmisión de diferentes datos en el mismo número de secuencia de subflujo, si hay que retransmitir se hace en el subflujo TCP original.
- ***Firewalls*** : estos dispositivos se describen en el RFC 2979 [53] y pueden cambiar el número de secuencia inicial o ISN de una conexión y no permitir muchas conexiones entrantes. Por ello, MPTCP utiliza números de secuencia relativos en asignaciones de secuencia de datos y además admite la señalización de direcciones (ADD_ADDR) para que un *host* pueda establecer una conexión adicional con otro que se encuentre detrás de un *firewall* o NAT.
- ***Middleboxes* a nivel de aplicación** : pueden alterar la carga útil dentro de un subflujo. Si pasa esto, como este protocolo detecta cambios en la carga útil utilizando el *Checksum* de DSS, puede cerrar los subflujos afectados y en caso de que no quede ninguno sin afectar, volver a TCP.
- ***Intrusion Detection System/Intrusion Prevention System* (IDS/IPS)** : un análisis en profundidad de estos sistemas

se llevó a cabo por la Universidad Piloto de Colombia [54]. Estos sistemas buscan patrones anómalos en los paquetes para intentar evitar amenazas en la red. MPTCP puede necesitar utilizar rutas adicionales y estos sistemas deberán leer *tokens* MPTCP para vincular datos de múltiples subflujos y mantener una visión de todo el tráfico. Si esto no ocurre, un sistema IDS/IPS podría tener una vista incompleta del tráfico y emitir falsos positivos y falsos negativos, reduciendo la eficiencia del protocolo.

Como se ha ido observando, cada una de las opciones MPTCP mencionadas corresponde con un valor asignado por IANA entre 0x0 y 0x7 en el campo *Subtype* de las opciones de TCP. El valor 0xf está reservado para uso privado en bancos de pruebas controlados, y los valores de 0x8 a 0xe están actualmente sin asignar.

3.2.4. MPTCP Versión 1

En marzo de 2020, el protocolo MPTCP pasó a una nueva y, por ahora última versión, la versión 1. Está especificada en el RFC 8684 [3]:

Motivación

Surge como intento de solucionar algunos de los problemas localizados en la versión 0, sobre todo, problemas de seguridad. Estos se especifican en el RFC 6181 [55], se complementan en el RFC 7430 [56], y se resumen en los que siguen:

- **Ataque ADD_ADDR**: en él, el atacante secuestra una sesión MPTCP y realiza un ataque *Man in the Middle* obteniendo toda la información que los participantes de la sesión se estén enviando. Se supone que el atacante conoce las direcciones IP y puertos de ambos extremos, un número de secuencia válido para el subflujo, un número de secuencia ACK también válido y un *Address* ID de dicha sesión. Los pasos son:

  - Los participantes de la sesión son *host* A y *host* B. El atacante es *host* C.

  - *Host* C envía un mensaje ADD_ADDR a *host* B con origen IP_A/P_A, destino IP_B/P_B, indicador ACK y número de secuencia válidos (anunciando como dirección adicional IP_C).

  - *Host* B manda un mensaje SYN+MP_JOIN a *host* C con origen IP_B/P'_B y destino IP_C/P'_A (conteniendo el *token* T_A y un *nonce* aleatorio). Nótese que P'_A y P'_B no tienen por qué ser iguales a P_A y P_B.

  - *Host* C reenvía dicho mensaje a *host* A, cambiando la dirección origen por IP_C y la de destino por IP_A; así suplanta a *host* B.

- *Host* A responde a *host* C con un SYN/ACK +MP_JOIN (conteniendo el HMAC calculado). *Host* C recibe el mensaje, lo modifica haciéndose pasar por *host* A y lo reenvía a *host* B.

- *Host* B recibe el mensaje, verifica el HMAC y manda el ACK+MP_JOIN a *host* C conteniendo su HMAC. Este modifica las direcciones origen y destino y lo reenvía a *host* A.

- Así, se crea el nuevo subflujo y *host* C ya está en mitad de la comunicación (*MitM*).

Para intentar evitar este ataque se puede poner como obligatorio el envío del *token* de la conexión en el mensaje ADD_ADDR (lo que obliga al atacante a espiarlo anteriormente), o incluir el HMAC (haciendo que éste tenga que conocer las claves de emisor y receptor, ya que la clave en el HMAC es una concatenación de ambas).

- **Ataque *Denial Of Service* (DoS) en MP_JOIN** : cuando un extremo de una comunicación quiere añadir un nuevo subflujo, manda un mensaje SYN+MP_JOIN al otro extremo; este se queda en un estado con la conexión semiabierta ya que en este mensaje se manda su *token* y un *nonce* que no se retransmite en el tercer ACK de los apretones de manos. El atacante, conociendo previamente el *token* del receptor válido, manda muchos mensajes SYN+MP_JOIN desde diferente IP origen a las direcciones que están en desuso para agotar el número de conexiones de este tipo que pueden albergar los extremos y así dejarlos sin posibilidad de crear nuevos subflujos.
  Como mejora, en el mensaje ACK final podría ir incluido el *token* y el *nonce* generado para el envío del SYN+MP_JOIN, haciendo que el receptor pudiese generar su propio *nonce* aleatorio y responder al SYN+MP_JOIN sin necesidad de crear un estado intermedio y quedarse con la conexión semiabierta, dándole la posibilidad al atacante de mandarle todos los mensajes y dejarlo bloqueado.

  Esta situación puede dar lugar a un doble ataque, produciendo además una inundación SYN (este ataque se define en el RFC 4987 [57]). Antes de mandar muchos mensajes SYN+MP_JOIN, el atacante puede enviar un SYN regular al *host* (servidor), creando un estado en ambos. Seguidamente, empieza a enviar mensajes SYN+MP_JOIN con diferentes *sockets* origen como se ha explicado, sin tener que crear en su lado ningún estado más, pero sí creándolo en el *host* para dichos mensajes. Así, el atacante solo tiene un coste asociado al estado creado por el SYN inicial, mientras que el *host* tiene el coste del estado SYN inicial junto con los estados asociados a todos los mensajes SYN+MP_JOIN recibidos, y por ello, puede que también tenga sus recursos agotados.
  Como soluciones, estarían por utilizar *cookies* SYN, que permitiesen

procesar el mensaje sin necesidad de crear un nuevo estado, limitar el número de subflujos semiabiertos y/o adoptar la solución sobre los ACK presentada antes.

- **Espionaje en el *Three-way Handshake* inicial** : el atacante intercepta el apretón de manos inicial que se produce en el establecimiento de conexión MPTCP entre dos *hosts*, y consigue las claves de ambos extremos (ya que en este momento se mandan en texto plano).
  Como principales soluciones se encuentran el uso de cadenas *hash* [58], de direcciones generadas criptográficamente [59], el uso de *tcpcrypt*, [60] y el de *Domain Name System Security Extensions* (DNSSEC) [61]. Todas estas soluciones no son motivo de estudio en este trabajo, por lo que en la bibliografía se dejan mencionadas en caso de que se desee profundizar más sobre ellas.
- **Ataque SYN/JOIN** : en este ataque, el delincuente cambia la dirección origen de los paquetes SYN/JOIN. Esto es difícil de resolver ya que MPTCP debe poder trabajar a través de NAT y, por tanto, no debe depender de asegurar la dirección origen. Un posible remedio es proteger la carga útil intercambiada. Así, aunque el atacante cambie la IP origen, si no es la información que se espera y no concuerda con las características, se descarta.

Una vez explicado el motivo por el que surge una nueva versión del protocolo, se procede a explicar el funcionamiento de esta actualización. Para evitar repeticiones, se exponen los procesos en los que se han añadido cambios con respecto a la versión ya explicada en el apartado 3.2.3, y aquellos que se han añadido nuevos.

Establecimiento de conexión

Uno de los procesos que se ve ligeramente modificado respecto a la versión anterior tiene que ver con la información intercambiada a la hora de establecer la conexión MPTCP. Al igual que en la versión 0, se hace un intercambio SYN, SYN/ACK, ACK junto con la opción MP_CAPABLE. Ahora el formato de esta opción sigue una estructura diferente, la cual se observa en la figura 3.24.

El primer cambio se puede ver en que ahora el campo ***Version*** irá con un 1; además, el campo ***Data-Level Length*** y ***Checksum*** se incorporan a la opción en caso de que se envíen datos junto con ella, y por último, no siempre van a ir incluidas las **claves**. Otra diferencia a tener en cuenta está en los *flags*: **A** y **B** se mantienen con la misma función, pero el *flag* **C** ya no es utilizado junto con los *flags* **D**-**H** para negociación de algoritmos criptográficos, sino que ahora tiene valor 1 cuando el remitente quiere indicar que no aceptará subflujos MPTCP adicionales a la dirección y puerto origen.

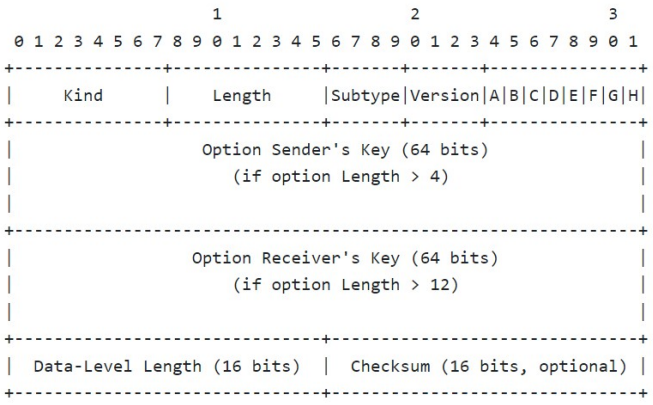


Figura 3.24: Opción MP_CAPABLE en MPTCP Versión 1 [3].

En caso de que un *host* que quiere iniciar la conexión admita ambas versiones de MPTCP, deberá utilizar la versión más alta, de forma que el receptor responda con la que desea utilizar (esta podrá ser igual o inferior a la indicada por el otro extremo). En este tema hay cierto problema ya que un oyente que funcione con la versión 0 del protocolo, estará esperando un SYN con un MP_CAPABLE que incluya la clave del remitente; sin embargo, si este remitente ya ha optado por emplear la versión 1 puede que no se incluya, en cuyo caso el oyente ignorará el mensaje y responderá con un SYN/ACK que no incluye el MP_CAPABLE. Es entonces, cuando el iniciador puede optar por pasar a TCP normal o tratar de volver a iniciar la conexión usando en este caso la versión 0 desde el principio. Normalmente, los *hosts* suelen guardar información sobre qué versión soportan los otros *hosts* para futuras conexiones.

Otra modificación que se puede observar tiene que ver con el caso en el que el iniciador de la conexión desee enviar datos y los tenga preparados en el momento de hacer el establecimiento de conexión. En dicho caso, este puede enviarlos junto con el mensaje ACK+MP_CAPABLE en el *three-way handshake* inicial. Así, el intercambio de mensajes sería como se muestra en la figura 3.25:

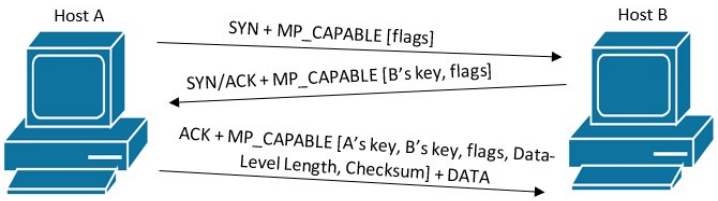


Figura 3.25: Caso 1 de Establecimiento de Conexión en MPTCP Versión 1.

En caso de que no haya datos para enviar inmediatamente y se envíen después, el remitente mandará el último ACK del apretón de manos con la opción MP_CAPABLE al igual que en la versión 0, y posteriormente enviará otro ACK con MP_CAPABLE incluyendo tanto el *Data-Level Lenght* como el *Checksum*, además de los datos. La figura 3.26 refleja esta secuencia:

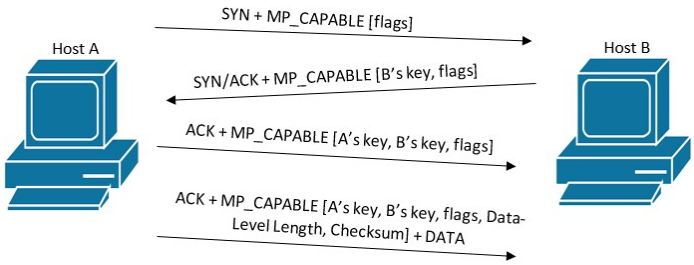


Figura 3.26: Caso 2 de Establecimiento de Conexión en MPTCP Versión 1.

Una última situación se da cuando es el extremo no iniciador de la conexión el que desea enviar los datos primero. En ese supuesto, el intercambio de mensajes se muestra en la figura 3.27. Se observa cómo el receptor responde al tercer ACK del *handshake* con un ACK+MP_CAPABLE+DATA, asegurando la recepción de estos datos con una opción *Data Sequence Signal* (DSS).

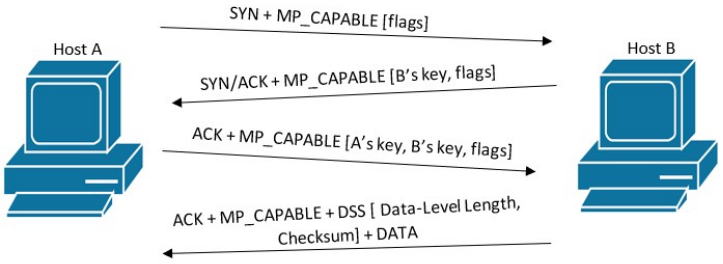


Figura 3.27: Caso 3 de Establecimiento de Conexión en MPTCP Versión 1.

Al igual que en la versión 0, las claves se envían en claro y se utilizan para generar *tokens*. Cuando alguno de los mensajes del apretón de manos no contenga esta opción, se volverá a TCP normal. Todas las demás características se mantienen.

Gestión de Subflujos

El proceso por el que se gestionan los subflujos es igual que en la versión anterior (los mensajes son los mismos y las reacciones ante los posibles casos

de fallo también). Solamente hay un detalle importante, y es que ahora los *tokens* que se emplean para autenticación no se generan con el algoritmo SHA1 sino con el algoritmo SHA256.

- Anuncio y Eliminación de Direcciones

Al igual que para la versión anterior, se emplea la opción ADD_ADDR, pero ahora su estructura cambia de forma ligera:

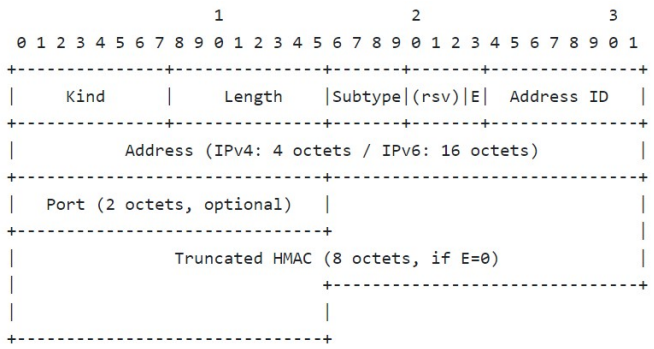


Figura 3.28: Opción ADD_ADDR en MPTCP Versión 1 [3].

Como se puede observar, frente a la figura 3.13, en la estructura del ADD_ADDR mostrada en la figura 3.28 aparecen 2 campos nuevos:

- Un campo **HMAC** truncado y calculado con el *hash* SHA256 (la clave es la concatenación de ambas claves, y el mensaje es el ID de dirección y la IP y puerto que preceden a éste en la opción). El objetivo de esta opción es intentar evitar que entidades maliciosas creen nuevos subflujos mediante esta opción con el fin de secuestrar una conexión (si un *host* recibe una opción ADD_ADDR con un HMAC que no reconoce, esta petición se ignora).
- La bandera o *flag* **E** se emplea para confirmaciones. Cuando un *host* reciba un mensaje ADD_ADDR con E=0, devolverá el mismo mensaje con E=1 y sin incluir el HMAC, indicando así su correcta recepción (si E se devolviese a 0, se estaría indicando al remitente que la opción debe retransmitirse).

El intercambio de mensajes es el que se refleja en la figura 3.29 (se añade la dirección IP_{A2}):

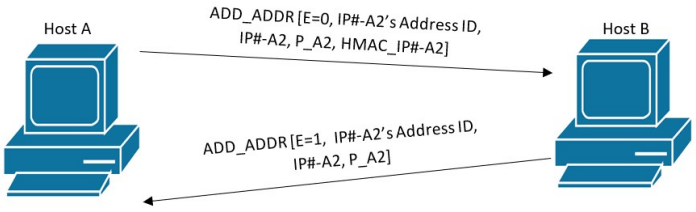


Figura 3.29: Anuncio de una Nueva Dirección en MPTCP Versión 1.

Al igual que la información sobre el número de versión soportado por cada *host*, puede ser almacenado por los demás; las direcciones disponibles de estos también se pueden guardar en caché para su uso futuro.

En el caso de querer eliminar una dirección, se utiliza la opción REMOVE_ADDR con la misma estructura e intercambio de mensajes que en la versión 0.

Fallback **o Retirada**

Los motivos por los que una implementación debe dejar MPTCP y volver a TCP regular son los mismos en ambas versiones, salvo por una excepción. Esta nueva versión sí tiene en cuenta cuando un subflujo se redirige y pasa por algún *middlebox* que causa problemas, no permitiendo las opciones MPTCP. Cuando ocurre esto, el subflujo se cierra con un RST que incluye la opción MP_TCPRST (véase la sección 3.2.4) con un código motivo de "interferencia de *middlebox*".

Otros Procesos

La transmisión de los datos, cierre de conexión, control de flujo, de congestión, de retransmisiones, gestión de fallos e interacciones con *middleboxes*, no sufren variación. El *path management* y *scheduling* también se presentan en esta versión, pero la forma de utilizarlos (en Linux) sí cambia y se explicará en el capítulo 5. Otros procesos añadidos o modificados son:

- ***FastClose*** **o Cierre Rápido**: la opción de cierre rápido MP_FASTCLOSE tiene la misma estructura que antes (figura 3.21), pero ahora no solo se puede llevar a cabo en los mensajes ACK, sino también en los RST:

  1. Opción RST: el *host* remitente envía un RST con la opción MP_FASTCLOSE en todos los subflujos, que a su vez contiene la clave del *host* receptor. Este, al recibir el mensaje, elimina todos los subflujos con un TCP RST. La conexión termina inmediatamente.
  2. Opción ACK: igual que en la versión anterior, pero ahora, si después de las 3 retransmisiones del MP_FASTCLOSE el *host* remi-

tente no recibe un TCP RST como respuesta, envía él mismo un TCP RST con la opción MP_FASTCLOSE.

- **Prioridad de Rutas/Política de Subflujo**: se utiliza la opción MP_PRIO para configurar la prioridad de un subflujo mediante el indicador **B**:

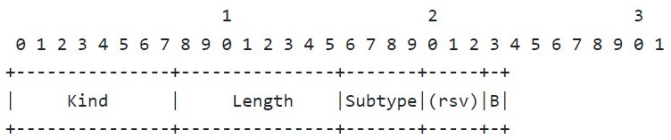


Figura 3.30: Opción MP_PRIO en MPTCP Versión 1 [3].

Como se observa en la figura 3.30, la opción ya no presenta el campo *Address* ID, por lo que se aplica únicamente al subflujo donde se envía. Además, esta opción se suele usar para poner un subflujo que se va a eliminar con baja prioridad, de modo que al eliminarlo con REMOVE_ADDR, no se esté usando y la eliminación sea más limpia.

- **Restablecimiento de Subflujo**: en esta nueva versión, a la hora de terminar un subflujo o querer que uno nuevo no se establezca, se usa también el TCP RST, pero ahora puede incluir una opción llamada MP_TCPRST que indique al receptor de ésta por qué se cierra o rechaza un determinado subflujo. Así, la mayoría de RST que se han mencionado en esta especificación irían acompañados de esta opción. El conjunto de campos que presenta ésta aparece en la figura 3.31:

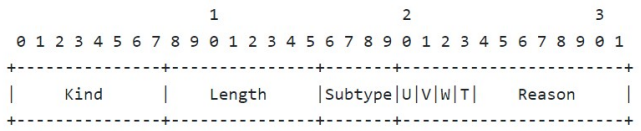


Figura 3.31: Opción MP_TCPRST en MPTCP Versión 1 [3].

Entre los campos relevantes se encuentran los siguientes:
- Los *flags* **T**,**U**,**V** y **W** : los tres últimos no se utilizan y están reservados para uso futuro, se ponen a 0 y el receptor los ignora. El *flag* **T** puede estar a 1 si la condición de error es temporal y el receptor puede intentar restablecer el subflujo; o permanente (T=0), en cuyo caso el receptor del *reset* no debe intentar restablecer el subflujo.
- Campo ***Reason***: campo de 8 bits que indica el motivo por el que se niega el subflujo. Este pueden ser: *0x00* (Error no especificado), *0x01* (Error específico de MPTCP), *0x02* (Falta de recursos), *0x03* (Prohibido administrativamente), *0x04* (Demasiados datos pendientes), *0x05* (Rendimiento inaceptable) y *0x06* (Interferencia con *middlebox*).

Por último, cabe destacar que al igual que se mencionó en la versión 0, cada una de las opciones MPTCP tiene un determinado valor asignado por IANA al campo *Subtype*. Los valores indicados en cada opción no cambian de una versión a otra.

Aquí se termina la exposición de los principales conceptos y procesos que se llevan a cabo en el protocolo *Multipath TCP* para sus dos versiones. Una vez adquiridos estos conocimientos teóricos, se da paso a la realización de una breve descripción sobre la implementación que se va a llevar a cabo para este protocolo, que como ya se mencionó en el capítulo 2, es la más extendida y se conoce como el *kernel* de Linux.

3.3. MPTCP en el *Kernel* de Linux

El autor de la implementación y arquitectura inicial de *Multipath TCP* sobre el *kernel* de Linux fue Sébastien Barré en el año 2009 basándose en su implementación de shim6 (*multihoming* por medio de direcciones IPv6 múltiples) [62]. Hoy en día, hay dos proyectos del *kernel* de Linux activos:

- ***Out-of-tree*** : está diseñado para tener buen rendimiento con MPTCP, pero impacta en TCP normal. Además, su mantenimiento es más complejo, soporta la especificación MPTCP versión 0 [2], y la versión 1 [3] desde la v0.96. Se puede usar con versiones de *kernel Long Term Support* (LTS) desde los más antiguos hasta la v5.5.

  Este proyecto se presenta como un parche para el *kernel* de Linux (también es posible obtener el *kernel* ya parcheado y compilarlo manualmente) y tiene su propio sitio web [29] donde se puede consultar desde cómo instalar el protocolo, hasta como configurarlo o hacer el enrutamiento necesario en los dispositivos para que funcione. Los desarrolladores actuales son Christoph Paasch, Fabien Duchêne y Gregory Detal.

- ***Upstream*** : desde la v5.6 del *kernel* oficial de Linux, MPTCP versión 1 viene soportado por defecto; sin embargo, este proyecto, alojado en GitHub, se encarga de ampliar los *kernels* para que además de tener el protocolo soportado, añadan funcionalidades más avanzadas.

Cabe destacar que MPTCP *Upstream* (MPTCP v1) no es compatible con MPTCP versión 0 y además, es la que hoy en día está en desarrollo. Por ello, seguidamente se comentan una serie de aspectos importantes del proyecto *Upstream* [30], analizados con más detenimiento en el artículo [63], redactado por los mismos desarrolladores del proyecto:

Como se ha mencionado, el *kernel* v5.6 fue la primera versión que incluyó

código MPTCP admitiendo únicamente conexiones de subflujo único (esto no traía ventajas con respecto a TCP normal), posteriormente apareció el *kernel* v5.7 con múltiples subflujos controlados por un gestor de rutas o *path manager* (su función se describió en el apartado *Otros Procesos* de la sección 3.2.3). Este fue el paso más grande de una versión a otra; a partir de aquí, cada nueva versión del *kernel* ha ido incluyendo más funcionalidades MPTCP mejoradas hasta día de hoy, con la versión v6.4 en pleno desarrollo. En todo este proceso, los diseñadores han ido cumpliendo una serie de pautas para intentar no salirse de lo establecido:

1. Mantener el rendimiento y fiabilidad de TCP. MPTCP se basa en él y lleva a cabo una serie de cambios en su estructura; aun así, TCP debe seguir funcionando plenamente.

2. Las aplicaciones que, por defecto, usan TCP, deben seguir haciéndolo independientemente de la incorporación de MPTCP.

3. MPTCP debe ser optativo con uso en caso de que se solicite explícitamente.

4. Ciertas características del protocolo son opcionales, por lo que se van implementando por lotes para poder ir probando su funcionalidad.

Además, para comprobar el correcto funcionamiento del protocolo, este grupo de investigadores ha incorporado una serie de herramientas de prueba [64]. Con estas se puede comprobar si las funciones que van implementando poco a poco, dan buenos resultados.

3.3.1. Plan de Trabajo en Desarrollo

En esta última sección se describen algunas tareas que se persiguen hoy en día en el proyecto *Upstream*.

Gestor de Rutas / *Path Manager*

Como ya se explicó en el apartado *Otros procesos* de la sección 3.2.3 de este proyecto, la gestión de los subflujos (creación, adición y eliminación) la lleva un gestor de rutas o *path manager*. Actualmente, este se puede gestionar directamente en el *kernel* mediante línea de comandos con *ip mptcp*. Para mejorar esto, se ha creado un demonio de espacio de usuario (*mptcpd*) de código abierto. Con él, el cliente (que suele ser el extremo con múltiples interfaces de red y posiblemente detrás de algún *middlebox*) puede configurar estos subflujos de una forma más rápida y simple. Este demonio sigue en desarrollo, aumentando su funcionalidad y eficacia.

Programador de Paquetes / *Packet Scheduler*

Existe un programador de paquetes que en función de unas determinadas

pautas elige el subflujo por el que se transmiten los datos. Se pueden transmitir por todos los subflujos a la vez, solo por el que tenga mejores condiciones, ir cambiando al que vaya mejor, etc. Dependiendo de esto, el *scheduler* puede ser de diferentes tipos.

En MPTCP *Upstream* el *scheduling* está desarrollo, pero actualmente se está optando por crear los diferentes *schedulers* a partir de módulos *Extended Berkeley Packet Filter* (eBPF) que se carguen en el *kernel*, para posteriormente elegir el que se quiera.

Sockets MPTCP

Actualmente, si se quiere que una conexión use *sockets* MPTCP de forma puntual, se puede usar la herramienta *mptcpize*, la cual fuerza su uso. Por otro lado, se puede crear una aplicación *.stap* que haga uso de IPROTO (protocolo de nivel inferior utilizado para enviar paquetes de datos a través de diferentes subcanales de una conexión MPTCP), cambiando IPROTO_TCP por IPROTO_MPTCP cuando se haga la llamada correspondiente al *kernel* (esto fuerza el uso de este tipo de *sockets* de forma permanente); sin embargo, esto no está todavía implementado al 100 %. Por ello, se está estudiando la posibilidad de establecer un nuevo punto de entrada eBPF en el *kernel*, el cual establezca siempre todos los *sockets* TCP a MPTCP, pero esto añadiría sobrecarga.

Break-Before-Make

Se plantea la posibilidad de tener escenarios en los que la conexión MPTCP entre los dispositivos se mantenga durante un tiempo limitado, aunque estos hayan cerrado todos los subflujos, con el fin de dar la posibilidad de restablecerlos de nuevo.

Linux Kernel Transport Layer Security (KTLS)

Por último, para mejorar la seguridad de MPTCP y añadir funcionalidades del protocolo *Transport Layer Security* (TLS) , se está evaluando cómo la utilización de KTLS afectaría a MPTCP, ya que al trabajar ambos en la capa de transporte, requeriría añadir una nueva capa *Upper-Layer Protocol* (ULP) para coordinar la utilización de ambos. Este plan presenta algunos desafíos técnicos, pero hay un modo de KTLS que parece factible. Aun así, no es objetivo de este proyecto la profundización de cómo funciona KTLS, por lo que en el estudio [65], publicado en la conferencia NetDev en 2018, se puede obtener más información si se requiere.

Una vez expuestos los conocimientos teóricos necesarios y una breve descripción de la implementación utilizada, se pasa a exponer tanto la planificación seguida, como el propio diseño e implementación de los escenarios.

Capítulo 4

Planificación y Costes

En el presente capítulo se exponen las diferentes fases que constituyen este proyecto, una descripción de cada una, y su duración. Además, se muestran los recursos empleados (humanos, hardware y software), junto con una estimación del coste asociado a cada uno. Finalmente, se concluye con una aproximación del presupuesto total del proyecto y un resumen con los costes de las modificaciones realizadas a posteriori.

4.1. Planificación Temporal

En esta primera sección se presentan las diferentes fases y tareas del proyecto; además de su duración y su posición a lo largo del tiempo disponible.

1. **Fase 1. Propuesta de Proyecto**
    - **Planteamiento del proyecto**: propuesta de trabajo y concertación entre tutor y alumno (12 días).
    - **Presentación y aceptación del proyecto**: presentación del proyecto y aceptación definitiva por parte de la escuela (12 días).
2. **Fase 2. Revisión Bibliográfica**
    - **Análisis del Estado del Arte**: estudio de los principales casos de uso del protocolo, y exposición de proyectos internacionales que hacen uso de él. Por último, se hace una breve descripción de las implementaciones que hay disponibles (12 días).
    - **Estudio de los conceptos teóricos fundamentales**: análisis y recopilación de información sobre el funcionamiento de MPTCP en todas sus versiones; así como una valoración de la implementación más extendida, y por ende, la que se va a usar (15 días).
    - **Referencia de la información plasmada**: recapitulación y verificación de las referencias y bibliografía empleada (6 días).

3. **Fase 3. Diseño e Implementación**

   - **Estudio de las posibles soluciones para implementar MPTCP versión 0**: diagnóstico de los requerimientos y procesos necesarios para poder diseñar e implementar una red con el protocolo en su versión inicial (14 días).
   - **Diseño e implementación de una red con MPTCP versión 0**: procedimiento de diseño e implementación de una red virtual con funcionamiento de MPTCP versión 0 entre sus dispositivos (14 días).
   - **Análisis de posibles soluciones para poner en marcha MPTCP versión 1**: estudio de las alternativas que hay actualmente para implementar el protocolo en su versión 1 (10 días).
   - **Diseño e implementación de una red con MPTCP versión 1**: proceso de diseño y configuración de una red virtual que albergue MPTCP versión 1 (20 días).

4. **Fase 4. Entorno de Experimentación y Pruebas**

   - **Establecimiento del entorno de experimentación y realización de pruebas con MPTCP versión 0**: se configura un entorno de experimentación con una serie de herramientas, con el fin de comprobar el funcionamiento y obtener datos sobre el rendimiento de la primera red virtual creada (28 días).
   - **Establecimiento del entorno de experimentación y realización de pruebas con MPTCP versión 1**: montaje de un segundo escenario de pruebas para verificar el correcto desempeño del protocolo en su última versión, y comparar rendimientos entre ambas especificaciones (28 días).

5. **Fase 5. Escritura y Entrega de la Memoria**

   - **Redacción de la memoria**: esta tarea empieza al mismo tiempo que la fase de pruebas y en su desarrollo se proceden a exponer de forma lógica y coherente todos los conocimientos y resultados obtenidos a lo largo de la elaboración del proyecto (71 días).
   - **Entrega de la memoria**: solicitud de evaluación del proyecto a través del correo y entrega de la memoria por Prado (11 días).
   - **Defensa del proyecto**: elaboración de la defensa del trabajo realizado y exposición ante el Tribunal escogido (6 días).

Para visualizar de manera más clara esta planificación, a continuación se muestra un diagrama de Gantt (figura 4.1) donde se pueden ver las diferentes fases y tareas, y su ocupación a lo largo del tiempo disponible:

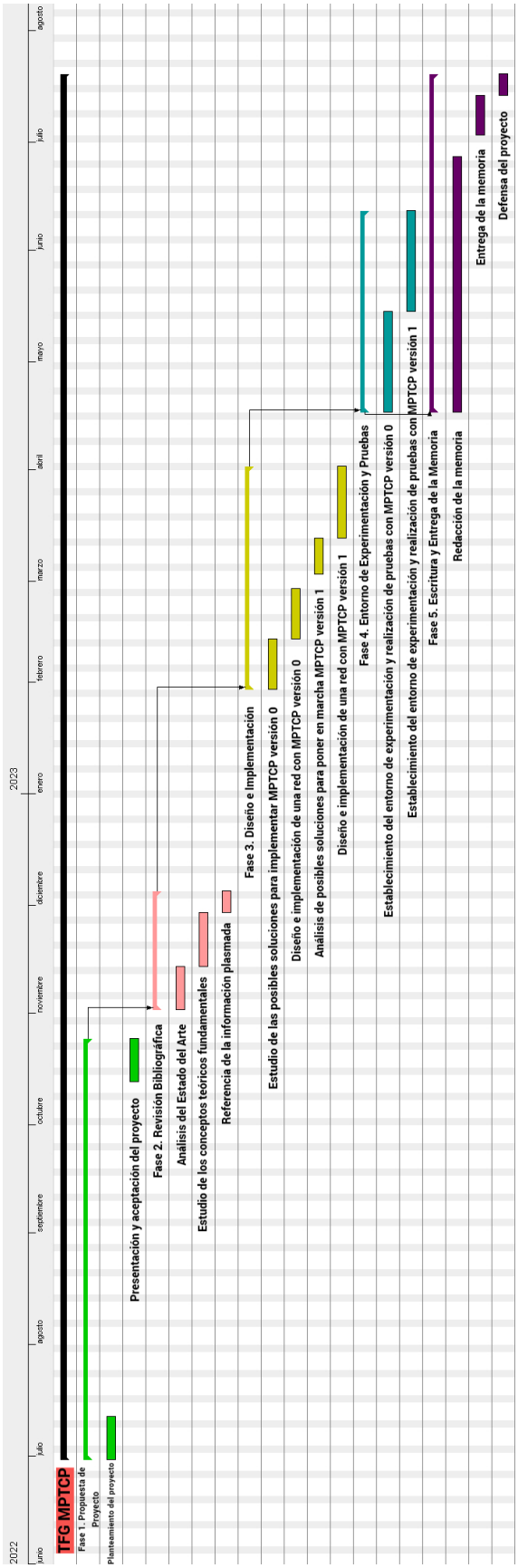


Figura 4.1: Diagrama de Gantt.

Por último y a modo de resumen, en la figura 4.2 se exponen de manera sistemática y temporal las diferentes actividades, junto con su fecha de inicio y fin y su duración en días, con el fin de dar una visión global final de la planificación de este proyecto.

Nombre	Fecha de inicio	Fecha de fin	Duración
TFG MPTCP	30/6/22	19/7/23	385
Fase 1. Propuesta de Proyecto	30/6/22	24/10/22	117
Planteamiento del proyecto	30/6/22	11/7/22	12
Presentación y aceptación del proyecto	13/10/22	24/10/22	12
Fase 2. Revisión Bibliográfica	2/11/22	4/12/22	33
Análisis del Estado del Arte	2/11/22	13/11/22	12
Estudio de los conceptos teóricos fundamentales	14/11/22	28/11/22	15
Referencia de la información plasmada	29/11/22	4/12/22	6
Fase 3. Diseño e Implementación	30/1/23	1/4/23	62
Estudio de las posibles soluciones para implementar MPTCP versión 0	30/1/23	12/2/23	14
Diseño e implementación de una red con MPTCP versión 0	13/2/23	26/2/23	14
Análisis de posibles soluciones para poner en marcha MPTCP versión 1	3/3/23	12/3/23	10
Diseño e implementación de una red con MPTCP versión 1	13/3/23	1/4/23	20
Fase 4. Entorno de Experimentación y Pruebas	17/4/23	11/6/23	56
Establecimiento del entorno de experimentación y realización de pruebas con MPTCP versión 0	17/4/23	14/5/23	28
Establecimiento del entorno de experimentación y realización de pruebas con MPTCP versión 1	15/5/23	11/6/23	28
Fase 5. Escritura y Entrega de la Memoria	17/4/23	19/7/23	94
Redacción de la memoria	17/4/23	26/6/23	71
Entrega de la memoria	3/7/23	13/7/23	11
Defensa del proyecto	14/7/23	19/7/23	6

Figura 4.2: Resumen de la Temporización de las Fases y Tareas del Proyecto.

4.2. Recursos y Coste

En esta sección se presentan los recursos humanos, hardware y software empleados en el proyecto, junto con el coste asociado a cada uno. Además, al final se hace una sumatoria de todos ellos para obtener el coste total asumido.

4.2.1. Recursos Humanos

En este tipo de recursos se tienen en cuenta las personas que han trabajado en el proyecto (tutor/es y estudiante), y el tiempo de trabajo de cada uno. Las personas que han participado en su elaboración final son:

- **Jorge Navarro Ortiz**. Profesor Titular de Universidad en el Departamento de Teoría de la Señal, Telemática y Comunicaciones. Asume el cargo de tutor del proyecto.

- **Félix Delgado Ferro**. Investigador Predoctoral en *Wireless and Multimedia Networking Lab* - UGR. Asume el cargo de cotutor del proyecto.
- **Alejandra Oliver Boada**. Estudiante del Grado en Ingeniería de Tecnologías de Telecomunicación en la Universidad de Granada. Asume el cargo de autora del proyecto.

Las horas dedicadas por parte del tutor y cotutor corresponden a las tutorías realizadas con la estudiante, la búsqueda de información, y el seguimiento y repaso tanto del proyecto en sí, como de la redacción de esta memoria.

En cuanto al tiempo de trabajo dedicado por mi parte, la estudiante, a continuación se muestra una tabla (tabla 4.1) con las diferentes tareas realizadas y las horas que ha conllevado cada una, así como la suma del tiempo destinado a la totalidad del proyecto.

Tarea	**Horas Dedicadas**
Planteamiento del proyecto	3
Presentación y aceptación del proyecto	0
Análisis del Estado del Arte	40
Estudio de los conceptos teóricos fundamentales	50
Referencia de la información plasmada	15
Estudio de las posibles soluciones para implementar MPTCP versión 0	10
Diseño e implementación de una red con MPTCP versión 0	30
Análisis de posibles soluciones para poner en marcha MPTCP versión 1	20
Diseño e implementación de una red con MPTCP versión 1	60
Establecimiento del entorno de experimentación y realización de pruebas con MPTCP versión 0	30
Establecimiento del entorno de experimentación y realización de pruebas con MPTCP versión 1	30
Redacción de la memoria	100
Entrega de la memoria	4
Defensa del proyecto	20
TOTAL: 412 horas	

Tabla 4.1: Tabla Resumen de Horas Dedicadas a las Diferentes Tareas.

En la tabla 4.2 se muestra un resumen del tiempo trabajado de todos los participantes, junto con el coste/hora (de acuerdo a su ocupación) de cada uno, y el presupuesto total humano.

Persona	Horas Trabajadas	Precio/hora (€)	Coste Total (€)
Jorge Navarro Ortiz	20	50	1000
Félix Delgado Ferro	20	35	700
Alejandra Oliver Boada	412	20	8240
TOTAL: 9940 €			

Tabla 4.2: Coste Total de Recursos Humanos.

4.2.2. Recursos Hardware

La única herramienta hardware utilizada es un portátil de uso personal modelo ASUS ZenBook 14, cuyas características se observan en la tabla 4.3:

Modelo	ASUS ZenBook 14
Procesador	AMD Ryzen 7 5700U (8 núcleos-16 hilos/4.3 GHz/12 MB caché)
Tarjeta Gráfica	AMD Radeon R7 Graphics Integrada
Almacenamiento	SSD M.2 NMVe PCIe 3.0 de 512 GB
Memoria RAM	16 GB SO-DIMM LPDDR4x
Sistema Operativo	*Windows* 11

Tabla 4.3: Características del Portátil Utilizado.

Ha sido importante disponer de un portátil con buenas características para poder albergar todos los programas y máquinas virtuales activas al mismo tiempo, y aun así tener un buen rendimiento y rapidez. Considerando que el tiempo de vida en el que un portátil funciona a pleno rendimiento es de 5 años, y que el uso que se le ha dado en el proyecto ha sido de 1 año, el coste proporcional a su precio inicial (620,13 €) se muestra en la tabla 4.4:

Elemento	Precio (€)	Vida Media	Uso	Coste Proporcional (€)
Portátil ASUS ZenBook 14	620,13	5 años	1 año	124,03

Tabla 4.4: Coste Total de Recursos Hardware.

4.2.3. Recursos Software

Los programas y herramientas software empleados son los siguientes:

- ***Oracle Virtual Machine* (VM) *VirtualBox* 7.0.6** [66]: software de virtualización multiplataforma para arquitecturas x86/amd64. Es de código abierto y permite la virtualización de casi cualquier sistema operativo, creando máquinas virtuales que pueden ser configuradas y gestionadas mediante una interfaz gráfica de usuario, y que cuentan con diversas funcionalidades.
  Esta plataforma se ha empleado para la creación de las redes virtuales del capítulo 5.

- ***Vagrant* 2.2.14** [67] : aplicación de línea de comandos que permite configurar instancias de máquinas virtuales (su sistema operativo, redes, redirección de puertos, usuarios o *scripts* de aprovisionamiento) mediante archivos de configuración llamados *Vagrantfile*. Este software trabaja normalmente con *VirtualBox*, por lo que en este proyecto es usado para generar entornos virtuales en esta plataforma.

- ***MobaXterm* 22.1** [68]: herramienta de escritorio remoto para *Windows* que incluye un servidor X11, el cual permite a las empresas conectarse y manejar tareas o aplicaciones desde computadoras remotas mediante la creación de diversas sesiones a distancia.
  Como el entorno gráfico de las máquinas virtuales en *VirtualBox* ocupa mucho espacio y ralentiza el rendimiento del equipo anfitrión, se usa este software para realizar una conexión *Secure Shell* (SSH) a estas máquinas y poder trabajar dentro de ellas con más ligereza.

- ***Grafana* 9.5** [69]: plataforma interactiva y de código abierto que permite la visualización de datos mediante gráficas y tablas. Se puede descargar en el equipo o acceder a ella a través del navegador en su versión *Grafana Cloud*. Ofrece una interfaz de usuario donde se pueden tener diversos paneles de control con gráficas que muestren lo que uno quiera. Esto facilita la interpretación y la comprensión de la información recabada mediante diferentes fuentes, generalmente bases de datos como Prometheus o *My Structured Query Language* (MySQL), entre otros.
  En este proyecto se usarán *Grafana* y *Prometheus* para elaborar gráficas que muestren los resultados extraídos de los escenarios virtuales.

- ***Matrix Laboratory (MatLab)* R2019** [70]: entorno de programación y de cálculo numérico que permite analizar datos, desarrollar algoritmos, crear modelos y generar gráficos integrados y personalizados para visualizar datos. Además, el lenguaje de programación usado es de alto nivel y propio, y también se denomina *MatLab*.
  Al igual que *Grafana*, se va a usar para procesar los datos obtenidos de las pruebas y elaborar gráficas que permitan analizarlos de manera más simple.

- ***Wireshark* 3.6.2** [71]: analizador de protocolos gratuito que permite capturar los paquetes que se intercambian en una red y analizarlos profundamente con el fin de observar los mensajes intercambiados en una comunicación de forma exhaustiva, y solucionar los problemas que haya en la red.
  Se utiliza para observar el intercambio de mensajes que ofrece MPTCP, una vez está en pleno funcionamiento en las redes virtuales creadas.

- **Sistema Operativo *Windows* 11 *Home*** [72]: sistema operativo desarrollado por *Microsoft* e instalado en el ordenador portátil que va a albergar las redes virtuales creadas mediante *VirtualBox*.
- **Sistema Operativo *Ubuntu* 18.04 LTS** [73]: sistema operativo que van a tener las máquinas virtuales en la implementación de MPTCP versión 0. Que una versión sea LTS quiere decir que recibe estabilidad, soporte extendido y actualizaciones de seguridad durante un período de tiempo mayor que las versiones regulares. La versión LTS más moderna es la 22.04, pero ésta se empleará en MPTCP versión 1, para que ambas implementaciones no sean con el mismo sistema operativo y haya variedad.
- **Sistema Operativo *Ubuntu* 22.04 LTS** [74]: sistema operativo que van a tener las máquinas virtuales en la implementación de MPTCP versión 1.
- ***Kernel* de Linux v5.4.144** [75]: el *kernel* es el elemento principal de los sistemas operativos Linux. Su función es comunicar el hardware con los procesos que se llevan a cabo mediante software, y gestionar los recursos de la mejor manera. La versión 5.4.144 es la que se va a usar en la implementación del primer escenario virtual, a la que se le aplicará el parche MPTCP versión 0.
- ***Kernel* de Linux v6.4-rc5** [76]: esta versión del *kernel* de Linux es hasta día de hoy la última que incluye soporte para MPTCP versión 1. En este Trabajo Fin de Grado se proporciona un enlace a un repositorio de GitHub creado por la estudiante donde se implementa MPTCP v1 en el *kernel* oficial de Linux (sin modificar); sin embargo, la experimentación y profundización del trabajo se va a llevar a cabo con la versión ampliada que ofrecen en GitHub los desarrolladores del proyecto *Upstream* [77]. Este *kernel* modificado (v6.4-rc5+) incluye, además de soporte a la versión 1 del protocolo, una serie de funcionalidades añadidas (como nuevos *schedulers*).
- ***Overleaf*** [78]: editor colaborativo y online de textos en formato LaTeX basado en la nube. Este editor es usado para la redacción, edición y publicación de documentos científicos, proporciona plantillas LaTeX de revistas oficiales, incluye un historial de todos los cambios para facilitar el manejo del documento al usuario y permite la colaboración en tiempo real de todos los editores, en caso de que haya más de uno. Este procesador es el que se emplea para la redacción de esta memoria.
- ***GanttProject* 3.2** [79]: programa multiplataforma de código abierto que permite la programación y gestión temporal de proyectos mediante la creación de gráficos Gantt de una forma sencilla. Este software de

escritorio ha sido utilizado para la elaboración del diagrama de Gantt mostrado en la figura 4.1.

- ***Zotero* 6.0.26** [80]: gestor de referencias bibliográficas gratuito y de código libre que se presenta dividido en dos partes, una extensión para los principales navegadores que permite guardar información de los sitios web de forma rápida, y una aplicación de escritorio donde recopilar, organizar, citar, sincronizar y gestionar todas las referencias almacenadas.
  Al permitir exportar las referencias en diferentes formatos, entre ellos BiblaTex, se ha utilizado en este proyecto para elaborar la bibliografía.

Casi todas las herramientas software empleadas son de código abierto, con el fin de reducir el coste del proyecto. Los dos únicos gastos adicionales que se han tenido son el de la licencia del sistema operativo *Windows* 11 *Home* que lleva instalado el ordenador portátil que se menciona en la sección 4.2.2, y el de la licencia de *MatLab*. El precio del portátil comentado en dicha sección incluía *Windows* 10 *Home* instalado, pero para poder amortizarlo mejor y durante más tiempo, su sistema operativo tuvo que actualizarse. El precio de esta licencia es de 145 €. En cuanto a *MatLab*, es un software comercial y por ello de pago, el precio de la licencia de estudiante con todo incluido es de 496 €(la licencia no tiene momento de expiración, es indefinida). El resumen de costes se observa en la tabla 4.5:

Software	**Precio (€)**
Oracle VM *VirtualBox* 7.0.6	0
Vagrant 2.2.14	0
MobaXterm 22.1	0
Grafana 9.5	0
MatLab 9.9	496
Wireshark 3.6.2	0
Sistema Operativo *Windows* 11 *Home*	145
Sistema Operativo *Ubuntu* 22.04 LTS	0
Kernel de Linux v5.5	0
Kernel de Linux v6.4-rc5 *Upstream*	0
Overleaf	0
GanttProject 3.2	0
Zotero 6.0.26	0
TOTAL: 641 €	

Tabla 4.5: Coste Total de Recursos Software.

4.2.4. Estimación de Costes

A continuación, se puede hacer una aproximación del coste total que ha tenido el proyecto. Esto se presenta en la tabla 4.6:

Recursos	Coste Total (€)
Humanos	10540
Software	641
Hardware	124,03
TOTAL: 11305.03 €	

Tabla 4.6: Coste Total del Proyecto.

4.2.5. Modificaciones en la Planificación Inicial

En el mes de Abril de 2023 se pensó en utilizar una serie de equipos disponibles en la Universidad para hacer una serie de pruebas MPTCP con instrumentación real, con el fin de que no fuese todo virtualizado y el proyecto se acercase un poco más a la realidad encontrada en proyectos internacionales. Inicialmente, las pruebas se iban a llevar a cabo tanto con MPTCP versión 0 como con versión 1, sin embargo, el tiempo era justo y se decidió montar el entorno de pruebas reales solo con la versión 0.

Los equipos utilizados tenían un periodo de amortización de unos 5 años, y fueron usados únicamente 1 semana. Aun así, al ser equipos de alto coste que ya estaban disponibles en la Universidad, no se han tenido en cuenta en el presupuesto final del proyecto, pero en la tabla 4.7 se deja un resumen de su coste en caso de interés.

Elementos	Precio (€)
2x *Next Unit of Computing* (NUC) (i7, 16 GB de *Random Access Memory* (RAM), 512 GB de *Solid State Drive* (SSD))	2x700=1400
Estación base de Amari SDR50 equivalente a una Amarisoft Callbox Classic	1500
Licencia AMARI LTE NW600 (licencia que deja usar un ancho de banda equivalente a 600 Mbps)	12000
PC	2000
Tarjeta Quectel RM500Q-GL + placa de desarrollo	500
Caja de Faraday LBX1000 Radio Frecuencia (RF) Shielded Test Enclosure	2500
Switch TRENDNET - TEG-S380 8 Port Gigabit	140
Un punto de acceso Wi-Fi 6 Xiaomi AX9000	300
TOTAL: 20340 €	

Tabla 4.7: Coste Total de los Equipos del Escenario Real.

El coste de este equipamiento, proporcional a un uso de 1 semana, sería de unos 78 €. Como se ha mencionado, estos equipos no se van a añadir al presupuesto final del proyecto, por lo que este se mantiene al mostrado en la tabla 4.6, unos 11305.03 €.

Capítulo 5

Diseño e Implementación

En el presente capítulo se exponen los pasos seguidos en el diseño, implementación y configuración de MPTCP en sus dos versiones.

5.1. Contexto Práctico

Antes de pasar a presentar el proceso seguido, se considera importante mostrar el entorno de configuración que se va a alcanzar al implementar MPTCP en el *kernel* de Linux, ya que ayuda a comprender mejor las configuraciones que se realizarán en este capítulo posteriormente.

En la práctica, el escenario que se presentó en la figura 3.7 se convierte en algo más parecido a lo mostrado en la figura 5.1, cuando la estructura lógica de una conexión MPTCP se integra en el *kernel* de Linux:

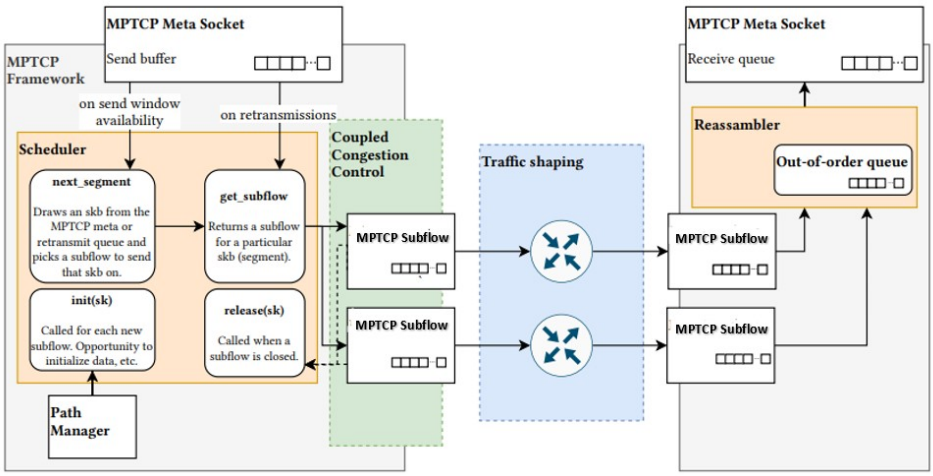


Figura 5.1: Entorno Práctico con MPTCP en el *Kernel* de Linux [81].

Como se observa en la figura 5.1, la configuración de una conexión MPTCP se produce de forma transparente a la aplicación. Ésta última interactúa con el *framework* MPTCP, en concreto, con el meta *socket* MPTCP, que es el que gestiona el *buffer* de envío a nivel de conexión y asigna a los datos su número de secuencia. Este *socket* se comunica con otro bloque del que ya se habló en la sección 3.2.3, el *scheduler*, enviándole información sobre los paquetes que están listos para enviarse o los que deben ser retransmitidos.

El *scheduler* determina qué datos se enviarán en cada subflujo dependiendo de una serie de condiciones. En Linux, se pueden implementar varios tipos de schedulers según la versión MPTCP utilizada (estos se comentan más adelante), pero en todos ellos se hace uso de varias funciones:

- La función *next_segment* selecciona un segmento y el subflujo por el que será enviado. A veces, el segmento enviado no se selecciona de la cola de envío, sino de la cola de retransmisión (cuando hay segmentos aquí, esta cola tiene mayor prioridad).
- La función *get_subflow* recibe un segmento de la cola de retransmisiones (en el *framework*), o de la función *next_segment* (en el *scheduler*), y le asigna el subflujo en el que será enviado.
- Las otras dos funciones del *scheduler*, *init* y *release*, son llamadas en la creación y cierre de subflujos.

Otro de los bloques que se comentaron en la sección 3.2.3 es el denominado *path manager*. Este gestor de rutas, encargado de establecer múltiples conexiones TCP sobre las que se transportan los datos, tiene una estructura modular, y si no es activado, el *host* no podrá crear nuevos subflujos ni anunciar direcciones IP alternativas a través de las opciones MP_JOIN y ADD_ADDR, como se vio en el capítulo 3. Para llevar a cabo esto, llama a la función *init* del *scheduler*. El *path manager*, implementado en Linux, también puede ser de varios tipos dependiendo de la versión del protocolo empleada, estas variantes se comentarán en las secciones posteriores.

Por otro lado, a la salida del *scheduler* se lleva a cabo el control de congestión. Dependiendo de la especificación que se esté llevando a cabo, se utilizará un algoritmo u otro. Los principales se comentaron en la sección 3.2.3.

Una vez que los paquetes salen del control de congestión, en caso de estar emisor y receptor en diferentes redes, pasarán por una serie de *routers* que pueden llevar a cabo acciones de modelado y conformación de tráfico. Al llegar al receptor, el *scheduler* reemsambla los paquetes de los diferentes subflujos y los coloca en la cola de recepción para posteriormente ser transmitidos a la aplicación.

Estos son los diferentes elementos que hay detrás de una conexión MPTCP sobre el *kernel* de Linux. Una vez expuestos de forma esquemática, se puede proceder con la explicación de cómo se implementa el protocolo (en sus dos versiones) en una red virtual haciendo uso de ello.

5.2. Marco de Configuración e Implementación de MPTCP Versión 0

A continuación, se describe el proceso seguido en la elaboración de un escenario virtual en el que se está utilizando MPTCP para el intercambio de datos.

5.2.1. Estudio de Soluciones

Como se presentó en la sección 3.3, para implementar MPTCP versión 0 en una máquina Linux, se utilizan los *kernels* hasta la versión v5.5. Siguiendo la web de la implementación [29], en concreto, el apartado *"How to install MPTCP"*, se puede observar cómo para implementar el protocolo en una máquina Linux se pueden seguir diversas alternativas. Entre las opciones que se presentan, se encuentran:

1. **Tomar el código fuente y compilarlo manualmente**: en el repositorio de GitHub [82], se encuentra la fuente (el *kernel*) para ser descargado, también se ofrece un enlace a los diferentes parches MPTCP [83]. Una vez se tiene descargado el código fuente del *kernel* y se le ha aplicado el parche MPTCP correspondiente, se debe acceder a su configuración y habilitar una serie de opciones [84] (activar el uso del protocolo, elegir el algoritmo de control de congestión, el *scheduler*, el *path manager*, o las políticas de *routing*). Configurado todo esto, se compila e instala el *kernel*, y se reinicia Linux. Lo siguiente sería configurar el *routing* del dispositivo (establecer sus interfaces, sus IP, puertos, pasarelas), para ello se pueden seguir los pasos mostrados en [85]. Con todo esto, se tendría un dispositivo Linux con funcionamiento de MPTCP versión 0.

2. **Instalar el *kernel* con soporte MPTCP de forma automática**: también se ofrece un enlace a un repositorio de GitHub que contiene los paquetes *.deb* que son necesarios instalar para tener el *kernel* de Linux que se desee con MPTCP versión 0.9x ya instalado (con el parche aplicado). De esta forma, para tener Linux funcionando con MPTCP, solo es necesario acceder al repositorio [86], descargar los paquetes dependiendo de la versión del *kernel* y MPTCP que se quiera instalar, instalarlos, y reiniciar la máquina; por último configurar el *routing* siguiendo los pasos de [85].

3. **Usar la herramienta *Vagrant* para obtener un *host* corriendo con MPTCP de forma directa**: si se desea que el dispositivo Linux sea una máquina virtual corriendo en un *host* anfitrión con otro sistema operativo, por ejemplo *Windows*, se puede usar la herramienta *Vagrant*. Como se mencionó en la sección 4.2.3, este software permite la creación de máquinas virtuales ya configuradas (soporte al protocolo, interfaces de red creadas y herramientas instaladas) en *VirtualBox* mediante el uso de un fichero de configuración llamado *Vagrantfile*, y una serie de comandos.

Estas soluciones se presentan adaptadas a diferentes distribuciones Linux como Debian, Fedora 19 al 21, *Community ENTerprise Linux Operating System* (CentOS) 7, Gentoo, ArchLinux, *Open Software- und System-Entwicklung* (OpenSUSE) o *Nix Operating System* (NixOS), entre otros. Así como para diferentes sistemas operativos basados en Linux como *Open Wireless Router* (OpenWRT) o Android, y plataformas o sistemas como Raspberry Pi, Amazon EC2 o PlanetLab. Para todos ellos hay enlaces a guías.

En este proyecto se escoge la tercera solución, uso de *Vagrant*, debido a:

- Para montar el escenario virtual se va a hacer uso de *VirtualBox*, por lo que la solución más rápida es mediante *Vagrant*, ya que crea máquinas de forma inmediata en él. Si se eligiese alguna de las otras dos soluciones, para crear una máquina habría primero que elegir la distribución que va a tener, crearla, y configurarla mediante la interfaz de usuario que ofrece *VirtualBox*. Con *Vagrant*, se ejecutan una serie de comandos en el *host* anfitrión y junto con un fichero de configuración, se crea una máquina con todas sus características definidas (S.O, *kernel*, memoria, *Central Processing Unit* (CPU), interfaces de red, herramientas de red) y con MPTCP instalado, en cuestión de minutos (el tiempo de creación depende de las capacidades del *host* anfitrión).

- Otra de las razones por las que esta herramienta ha sido elegida, tiene que ver con la reproducibilidad que ofrece. La idea es crear un escenario con dos máquinas que se comuniquen usando MPTCP versión 0, por lo que sería eficiente poder reproducir la misma máquina dos veces y después, simplemente cambiar el *routing* de la segunda.

- La última ventaja que ofrece esta solución es la portabilidad y facilidad de uso. *Vagrant* es compatible con múltiples proveedores de virtualización, además de *VirtualBox*, como *VMWare* o *Docker*. Esto permite utilizar la misma configuración en diferentes plataformas y sistemas operativos, lo que aporta gran portabilidad en el entorno de desarrollo. Esta característica se consideró importante ya que al principio del proyecto no se podía conocer el comportamiento que iba a tener *VirtualBox* en el *host* anfitrión, o si podía dar lugar a problemas; teniendo

visión a futuro, se tenía un resguardo o plan B donde intentar montar el escenario en caso de que la primera opción, *VirtualBox*, fallase.

5.2.2. Diseño de la Red

Como se ha mencionado en la sección 5.2.1, el objetivo es crear una red virtual con dos máquinas Linux, cada una con una interfaz a Internet, y con 3 interfaces a la misma red interna, que se comuniquen entre sí usando MPTCP versión 0. En la figura 5.2 se muestra un esquema de esta red:

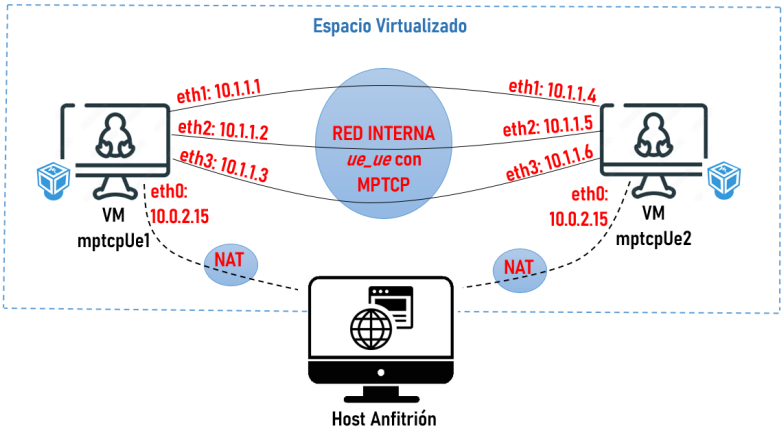


Figura 5.2: Escenario Virtual para MPTCP Versión 0.

Como se observa en la figura 5.2, cada máquina tiene 3 interfaces (*eth1*, *eth2* y *eth3*) a una misma red interna bajo el nombre de *ue_ue* (entre estas interfaces se establecerá MPTCP versión 0 en ambas máquinas). Esto se traduce en 3 adaptadores de red interna en la configuración de las máquinas en *VirtualBox*. A su vez, cada una tiene una interfaz *eth0* que se comunica con el *host* anfitrión mediante una red virtual, en *VirtualBox* se denomina adaptador NAT. La función de esta interfaz es comunicar la máquina con el *host* anfitrión y que éste haga una traducción de direcciones NAT entre la IP de la interfaz *eth0* (10.0.2.15) y una IP de la red física, proporcionando a la máquina conectividad con redes externas como Internet.
De esta forma, el anfitrión hace de intermediario enrutando el tráfico entre la máquina virtual y la red física, estableciendo así una comunicación bidireccional entre ellos y permitiéndole a la máquina virtual enviar y recibir paquetes de datos a través de la interfaz NAT como si estuviera conectada directamente a la red física.

El proceso de implementación y puesta en marcha de esta red mediante *Vagrant*, se describe seguidamente en la sección 5.2.3.

5.2.3. Creación del Escenario mediante *Vagrant*

Para realizar la implementación de la red vista en la figura 5.2 mediante la herramienta *Vagrant*, se va a hacer uso del repositorio de GitHub [27], el cual contiene los pasos y archivos de *Vagrant* necesarios para la creación del escenario virtual. Además, todos sus archivos están en lenguaje *Ruby*.

Lo primero que se debe hacer es descargar tanto el software de virtualización *VirtualBox*, como la herramienta *Vagrant*. Para ello se debe acceder a los enlaces proporcionados en la sección 4.2.3, correspondientes a los sitios web oficiales de ambos, y descargar e instalar las versiones necesarias en el *host* anfitrión (la versión mínima de cada uno se muestra en dicha sección).
Una vez que se tengan ambos instalados, se puede proceder a descargar el repositorio de GitHub [27], con el fin de tener en el *host* anfitrión los archivos que se van a necesitar en la ejecución de *Vagrant*. Siguiendo los pasos que se muestran, lo siguiente sería copiar una serie de credenciales SSH en uno de los directorios del repositorio descargado, para que una vez las máquinas estén creadas, se pueda descargar el repositorio dentro de ellas mediante una conexión SSH; sin embargo, esto ya no es necesario, ya que la descarga de éste se va a hacer de forma diferente (esto se muestra más adelante en esta descripción).

Como se mencionó en las secciones 4.2.3 y 5.2.1, para crear y configurar una máquina en *VirtualBox* mediante esta herramienta, se necesita un archivo de configuración denominado *Vagrantfile*. Pues bien, si se accede al lugar donde este repositorio se ha descargado en nuestro *host*, y se navega hasta el directorio *multitechnology_testbed_v0/vagrant*, se puede observar como hay un archivo denominado *Vagrantfile.2machines*; éste va a ser el archivo que configure el escenario virtual con las dos máquinas conectadas entre sí mediante MPTCP. Para poder utilizarlo, es necesario cambiarle el nombre a *Vagrantfile* (si tiene un nombre diferente, hay problemas).

Este archivo lleva a cabo una serie de tareas mediante las cuales se instalan y configuran las herramientas necesarias para la creación del escenario. Entre estas tareas se encuentran, la asignación del sistema operativo a ambas máquinas como *Ubuntu* 18.04 LTS, y la utilización del *kernel* 5.4.144 con parche MPTCP. La explicación del contenido del archivo, junto con algunos ficheros más que son utilizados por él, se lleva a cabo en el Apéndice A. Si se quiere profundizar en lo que se realiza dentro de estos archivos, se puede consultar allí.

Continuando con el proceso de instalación, cuando se ha cambiado el nombre de *Vagrantfile.2machines* a *Vagrantfile*, se debe acceder a la interfaz de línea de comandos del *host* anfitrión y ejecutar los siguientes dos comandos:

`1. sudo vagrant plugin install vagrant-reload`

Este comando instala el complemento de *Vagrant* llamado *"vagrant-reload"*, el cual permite recargar o reiniciar la máquina virtual si alguna configuración lo requiere (a veces se necesita hacer esto para que los cambios en una configuración se guarden). El archivo *Vagrantfile*, en la parte de aprovisionamiento, hace uso de esta recarga después de ejecutar algunos de los *scripts* descritos en el Apéndice A, por lo que es importante tener el complemento instalado para que todo funcione.
Después de esto, y como acción final, se debe ejecutar el siguiente comando:

`2. sudo vagrant up`

Éste verifica si existe la máquina o máquinas virtuales (como es este caso), que se especifican en el *Vagrantfile*. En caso de existir lo avisa, y en caso de no existir, crea y configura la/s máquina/s en *VirtualBox* siguiendo las directrices del *Vagrantfile*, y proporcionándoles además una interfaz de línea de comandos para poder comunicarnos con ellas.

Siguiendo estos pasos, se puede acceder a la interfaz de usuario de *VirtualBox* y verificar cómo se han obtenido dos máquinas como las de la figura 5.2.

A continuación, se debe descargar el software *MobaXterm* (en la sección 4.2.3 se proporciona el enlace para su descarga). Una vez se ha realizado la descarga del programa, dentro de él se deben crear dos sesiones SSH con puertos remotos 22, y puertos locales 12222 y 22222. Al crear ambas sesiones, se pide una contraseña, esta es *vagrant*. Como se observa en la figura 5.3, ya se está dentro de las máquinas virtuales *mptcpUe1* y *mptcpUe2*:

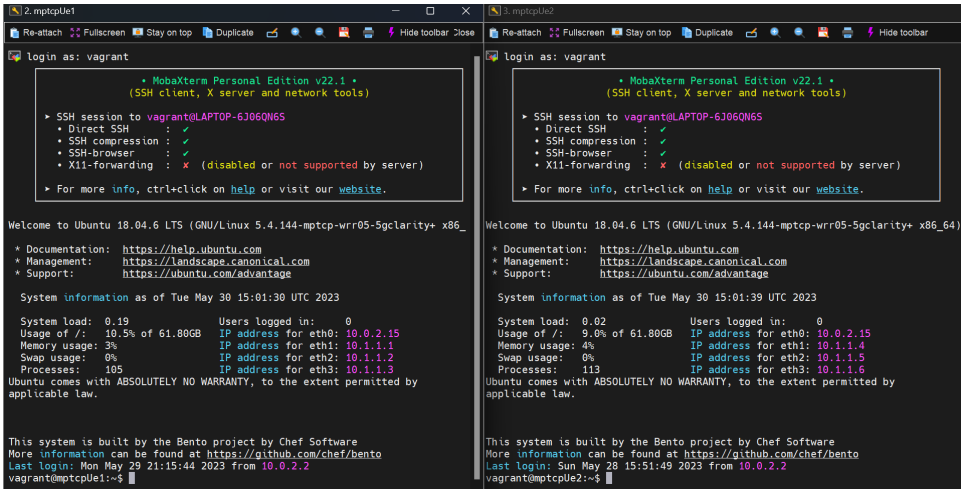


Figura 5.3: Conexión SSH a *mptcpUe1* y *mptcpUe2*.

Configuración del Sistema

Las direcciones IP de las interfaces dentro de la red interna *ue_ue* ya se definieron usando *Vagrant*, pero el *routing* no está hecho. Además, para que MPTCP funcione correctamente se debe elegir todavía el *scheduler*, *path manager* y control de congestión que se desea utilizar en el protocolo.

Para establecer todo ello, se usa un fichero que se encuentra en el repositorio que se está utilizando como guía [27]. Debido a su gran extensión, no se va a mostrar su contenido de manera explícita (si se quiere ver, se puede encontrar en la carpeta *mptcp_test* en el repositorio mencionado). Los comandos a ejecutar en cada máquina, haciendo uso de este fichero, son los siguientes:

mptcpUe1

```
1. cd free5gc/mptcp_test

2. ./set_MPTCP_parameters.sh -p <path manager> -s <scheduler> -c

<control congestion> -f if_names.txt.scenario1_same_network_UE1
```

mptcpUe2

```
1. cd free5gc/mptcp_test

2. ./set_MPTCP_parameters.sh -p <path manager> -s <scheduler> -c

<control congestion> -f if_names.txt.scenario1_same_network_UE2
```

Como se puede observar, se utiliza *set_MPTCP_parameters.sh*, que es un *script* de *shell* que realiza la asignación de direcciones IP, la configuración de reglas de *routing*, la configuración de parámetros MPTCP como el *scheduler* o el *path manager*, la creación de conexiones *openvpn* y opciones de depuración, etc. Para poder llevar a cabo esto, el archivo utiliza nombres de interfaz y otras configuraciones específicas que se proporcionan a través de opciones, estas opciones son los parámetros que se le pasan como entrada. Pueden ser varios:

- **-p <*path manager*>**: mediante esta opción se establece el tipo de *path manager* que se va a utilizar. Para esta versión del protocolo hay 4 tipos de gestor de rutas:
  - ***Default***: el *host* no anuncia nuevas direcciones ni inicia la creación de nuevos subflujos, pero sí acepta su creación por parte de otros *hosts*.

  - ***Fullmesh***: es el más usado. Con él, el *host* cliente anuncia sus direcciones IP al *host* servidor, y escucha las que son anunciadas por él. Además, el cliente es el único que añade o elimina direcciones según aumente o disminuya el número de interfaces, el otro dispositivo no crea los subflujos para evitar problemas con NAT o *firewalls*, como se explicó en la sección 3.2.3 (si éste tiene N direcciones y el servidor M, establece NxM subflujos, lo cual no siempre es eficiente).
  - ***Ndiffports***: con este gestor se crean subflujos a través del mismo par de direcciones IP, modificando solo el puerto de origen.
  - ***Binder***: es el *path manager* menos recomendado ya que no aprende de forma automática las direcciones IP de ambos extremos, y tampoco reacciona a los cambios en las interfaces. Como ventaja está que permite agregar múltiples pasarelas de Internet distribuidas geográficamente en redes comunitarias usando MPTCP, tunelización *openvpn* y *routing* de origen libre, evitando los efectos negativos de la reordenación de paquetes.

- **-s <*scheduler*>** : se establece el tipo de programador de paquetes que se va a emplear. Hay 3 tipos de *scheduler*:

  - ***Default***: es el más eficiente. Primero envía los datos por el subflujo con menor *Round Trip Time* (RTT), hasta que su ventana de congestión se llene, después envía por el que tenga el siguiente RTT más bajo, y así sucesivamente.
  - ***Round-Robin***: este planificador se suele utilizar en espacios de prueba académicos. Lo que hace es transmitir un paquete por subflujo de forma round-robin, pudiéndose establecer si se quiere que cuando la ventana de congestión de algún subflujo esté llena se salte al siguiente, o espere a que se vacíe para poder transmitir por él consiguiendo así un round-robin realista. Además, se puede configurar el número de segmentos que se envían de forma consecutiva, ya que por defecto es 1.
  - ***Redundant***: envía los mismos datos por todos los subflujos de forma redundante. Esto disminuye la latencia, pero requiere más ancho de banda.

- **-c <control congestion>**: con esta opción se está estableciendo el algoritmo de control de congestión usado. Se pueden usar algoritmos como OLIA, LIA, wVegas o BALIA, establecidos formalmente por el RFC 6824 [2] como algoritmos de control de congestión usados en MPTCP; o Reno, Cubic y mctcpdesync. Los cuatro primeros se describen en la sección 3.2.3, en cuanto a los demás:

  - **Reno** : se basa en el Reno de TCP, utilizando la detección de pérdidas de paquetes para ajustar la tasa de transmisión y controlar la congestión en los diferentes subflujos.
  - **Cubic** : se basa en el Cubic de TCP, utilizando una función cúbica para ajustar la ventana de congestión en función del tiempo que ha pasado desde la última congestión que se detectó.
  - **mctcpdesync** : con este algoritmo los subflujos van adaptando la carga y equilibrándose para evitar la congestión.

- **-f *if_names.txt.scenario1_same_network_UEX*** : con esta opción se está pasando como parámetro un archivo *.txt*, en caso de *mptcpUe1* se llama a *if_names.txt.scenario1_same_network_UE1*, y en *mptcpUe2* a *if_names.txt.scenario1_same_network_UE2*. Estos archivos contienen la IP de cada interfaz (según se vio en la figura 5.2) y la pasarela por defecto a la que se conectan (ambas máquinas tienen como pasarela la interfaz *eth1* de su otro extremo). Con estos parámetros, el *script set_MPTCP_parameters.sh* realiza el *routing* en cada máquina.

Además de los parámetros descritos, que son los básicos que se utilizan, se pueden añadir otros:

- **-C Y/N (CWND_LIMITED=Y/N)**: con el *round-robin scheduler*, -C Y indica que si un subflujo tiene la ventana de congestión llena, se salte al siguiente. Con -C N se espera a que se vacíe y se transmite.
- **-m**: si se quiere crear un espacio de nombres MPTCPns con interfaces virtuales.
- **-o *client/server***: para crear una conexión *openvpn* indicando si la entidad va a ser servidor o cliente.
- **-N IP**: para indicar la IP de la red *openvpn* en la que la máquina es servidor.
- **-S IP**: para indicar la IP de los servidores de la red *openvpn* de la que la máquina es cliente.
- **-d**: se utiliza para indicar el deseo de imprimir mensajes de depuración.

Una vez se han ejecutado dichos comandos en ambas máquinas, se puede observar el *routing* en la figura 5.4. En cada máquina se han creado 3 tablas de enrutamiento basadas en la IP de origen (*eth1*, *eth2* y *eth3*), cada tabla tiene dos entradas: la primera entrada indica que para el tráfico que salga de la interfaz *eth1* (tabla 1), *eth2* (tabla 2) o *eth3* (tabla 3), y que no tenga destino definido, la IP por defecto será la IP de la interfaz *eth1*

del otro extremo (10.1.1.1 para *mptcpUe2* y 10.1.1.4 para *mptcpUe1*). La segunda entrada indica que cualquier tráfico destinado a la red con el prefijo 10.1.1.0/24, debe enviarse directamente a través del dispositivo de red *eth1*, *eth2* o *eth3* (dependiendo de la tabla que se esté configurando).

Además, se comprueba que es similar a la que se muestra como ejemplo en el apartado *Configure Routing* de la web oficial de MPTCP versión 0 sobre el *kernel* de Linux [85].

```
vagrant@mptcpUe1:~$ ip rule show
0:      from all lookup local
32763:  from 10.1.1.3 lookup 3
32764:  from 10.1.1.2 lookup 2
32765:  from 10.1.1.1 lookup 1
32766:  from all lookup main
32767:  from all lookup default
vagrant@mptcpUe1:~$ ip route
default via 10.1.1.4 dev eth1
default via 10.0.2.2 dev eth0 proto dhcp src 10.0.2.15 metric 100
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
10.0.2.2 dev eth0 proto dhcp scope link src 10.0.2.15 metric 100
10.1.1.0/24 dev eth3 proto kernel scope link src 10.1.1.3
10.1.1.0/24 dev eth2 proto kernel scope link src 10.1.1.2
10.1.1.0/24 dev eth1 proto kernel scope link src 10.1.1.1
vagrant@mptcpUe1:~$ ip route show table 1
default via 10.1.1.4 dev eth1
10.1.1.0/24 dev eth1 scope link
vagrant@mptcpUe1:~$ ip route show table 2
default via 10.1.1.4 dev eth2
10.1.1.0/24 dev eth2 scope link
vagrant@mptcpUe1:~$ ip route show table 3
default via 10.1.1.4 dev eth3
10.1.1.0/24 dev eth3 scope link
vagrant@mptcpUe1:~$ ip route show table 4
```

```
vagrant@mptcpUe2:~/free5gc/mptcp_test$ ip rule
0:      from all lookup local
32763:  from 10.1.1.6 lookup 3
32764:  from 10.1.1.5 lookup 2
32765:  from 10.1.1.4 lookup 1
32766:  from all lookup main
32767:  from all lookup default
vagrant@mptcpUe2:~/free5gc/mptcp_test$ ip route
default via 10.1.1.1 dev eth1
default via 10.0.2.2 dev eth0 proto dhcp src 10.0.2.15 metric 100
10.0.2.0/24 dev eth0 proto kernel scope link src 10.0.2.15
10.0.2.2 dev eth0 proto dhcp scope link src 10.0.2.15 metric 100
10.1.1.0/24 dev eth2 proto kernel scope link src 10.1.1.5
10.1.1.0/24 dev eth3 proto kernel scope link src 10.1.1.6
10.1.1.0/24 dev eth1 proto kernel scope link src 10.1.1.4
vagrant@mptcpUe2:~/free5gc/mptcp_test$ ip route show table 1
default via 10.1.1.1 dev eth1
10.1.1.0/24 dev eth1 scope link
vagrant@mptcpUe2:~/free5gc/mptcp_test$ ip route show table 2
default via 10.1.1.1 dev eth2
10.1.1.0/24 dev eth2 scope link
vagrant@mptcpUe2:~/free5gc/mptcp_test$ ip route show table 3
default via 10.1.1.1 dev eth3
10.1.1.0/24 dev eth3 scope link
```

Figura 5.4: Configuración del *Routing* en Ambas Máquinas Virtuales (MPTCP versión 0).

Por otro lado, el *scheduler*, *path manager* y control de congestión son configurables mediante variables *sysctl*. Estas variables son parámetros de configuración del *kernel* de Linux que se pueden cambiar, modificando el comportamiento de ciertos elementos del sistema. Al pasarle un tipo de *scheduler*, *path manager* y control de congestión al archivo *set_MPTCP_parameters.sh*, éste asignó dichos valores a las variables *sysctl* correspondientes. Las asignaciones se pueden ver en la figura 5.5.

```
vagrant@mptcpUe1:~$ sudo sysctl net.mptcp
net.mptcp.mptcp_binder_gateways =
net.mptcp.mptcp_checksum = 1
net.mptcp.mptcp_debug = 0
net.mptcp.mptcp_enabled = 1
net.mptcp.mptcp_path_manager = fullmesh
net.mptcp.mptcp_scheduler = default
net.mptcp.mptcp_syn_retries = 3
net.mptcp.mptcp_version = 0
```

Figura 5.5: Configuración de Variables *sysctl net.mptcp*.

Además de las mencionadas, hay más variables relacionadas con la versión, la depuración o el *checksum*, que se pueden modificar. En el caso del control de congestión, su variable se denomina *net.ipv4.tcp_congestion_control*, por lo que con el comando que se muestra seguidamente se puede ver como

su valor es *olia*.

```
sudo sysctl net.ipv4
```

Se destaca también que se puede cambiar el valor de estas variables mediante el siguiente comando:

```
sysctl -w net.mptcp.[nombre de la variable] = [valor]
```

Verificación del Sistema

Las pruebas cambiando la configuración MPTCP se realizarán en el capítulo 6, pero a continuación se muestra una pequeña prueba inicial que corrobora el funcionamiento del sistema. Para ello, se ha mantenido la configuración MPTCP que se muestra en la figura 5.5 (el *path manager* se mantiene como *fullmesh*, asegurando el intento de creación de subflujos por parte de cada una de las interfaces de cada máquina con todas las de la otra, el *scheduler* como *default*, y el control de congestión se mantiene OLIA).

Con dicha configuración, y en el directorio *free5gc/mptcp_test*, se ejecuta:

mptcpUe1

```
./test_throughput_tcp_server.sh & ifstat
```

mptcpUe2

```
./test_throughput_tcp_client.sh -s 10.1.1.1 & ifstat
```

Se observa cómo se emplean dos archivos nuevos: *test_throughput_tcp_server.sh* y *test_throughput_tcp_client.sh*. Estos archivos también se pueden ver con detenimiento en el repositorio, pero básicamente realizan lo siguiente:

- *mptcpUe1* actúa como servidor, por lo que se utiliza la herramienta *iperf* con la opción -s para iniciar el servidor de prueba y ponerse en escucha de nuevas conexiones; además, se puede añadir la opción -f para indicar un fichero donde guardar los resultados. Cabe destacar que *iperf* es una herramienta para medir el rendimiento de la red tanto TCP como UDP.
- *mptcpUe2* actúa como cliente. Realiza una prueba de medición de ancho de banda en las interfaces activas mediante la herramienta *iperf*, introduciendo como parámetros de entrada: la dirección del servidor al que se conecta mediante la opción -s (en este caso la conexión se hace a la IP 10.1.1.1), la duración de la prueba de red mediante la opción -d (por defecto la duración se establece a 100 segundos), y por último el nombre del archivo para guardar los resultados usando la opción -f (este último parámetro es opcional).

Además, ambos comandos se ejecutan junto con la herramienta *ifstat*. Esta herramienta permite mostrar en línea de comandos las estadísticas de tráfico de red que se van obteniendo en tiempo real.

Si se pone *mptcpUe1* en modo escucha (servidor), y a continuación se hace la conexión de *mptcpUe2* (cliente) a su IP correspondiente, se obtiene la salida observada en la figura 5.6.

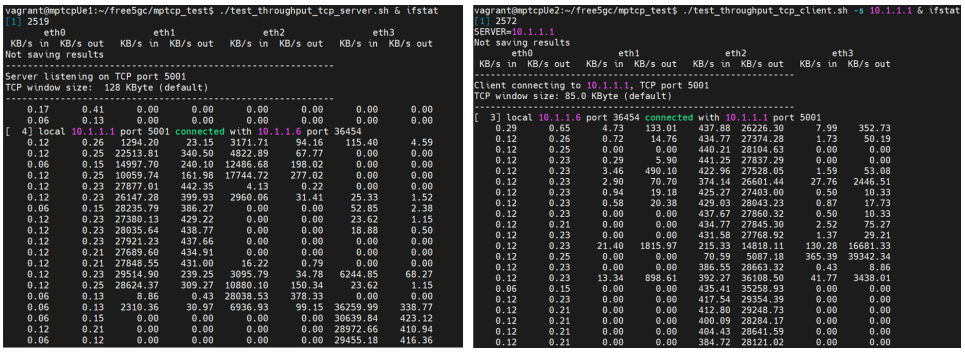


Figura 5.6: Prueba Inicial de MPTCP Versión 0.

Como se puede ver en la figura 5.6, utilizando el *scheduler default*, se mandan y reciben datos por el subflujo con el menor RTT hasta que su ventana de congestión esté llena, seguidamente, comenzará a transmitir en el subflujo con el siguiente RTT más bajo, y así sucesivamente. Además, el cliente es el que manda más volumen de datos, mientras que el servidor es el que recibe más datos y envía menos cantidad. El *throughput* que se alcanza, junto con los mensajes de señalización que se han intercambiado ambos extremos, se estudiarán con más detenimiento en el capítulo 6.

Un último concepto a mencionar es que las interfaces pueden configurarse para estar activas, inactivas, o en reserva o *backup*. Al poner una interfaz en este último estado (considerado solo en el *default scheduler*, en los demás se trata como activo (*on*)), los subflujos que se creen entorno a ella servirán de reserva; además, se puede establecer desde un inicio (opción MP_JOIN), o en mitad de la comunicación (opción MP_PRIO).

Se ha realizado una prueba para comprobar esto, mostrada en la figura 5.7. Se puede ver como, si la interfaz *eth1* se pone en modo reserva (color rojo), mientras que las demás funcionen, los subflujos asociados a ella no se usan. En caso de que la interfaz *eth2* se caiga (color verde), se sigue utilizando uno de los subflujos de *eth3*. Por último, si éste se cae (color azul), *eth1* se reactiva (color amarillo) salvando la conexión.

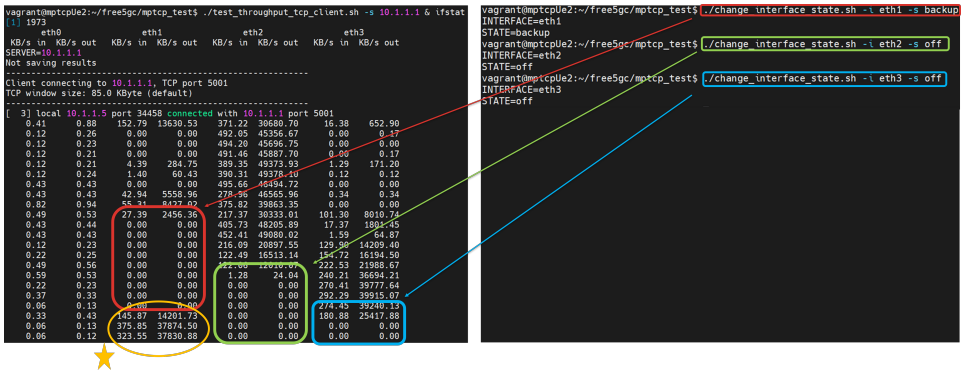


Figura 5.7: Subflujos de Reserva en una Conexión con MPTCP Versión 0.

Para cambiar el estado de las interfaces se ha usado un nuevo *script* bajo el nombre de *change_interface_state.sh*. Este se encuentra también en el repositorio, en la misma carpeta que los demás. Con él, se introducen dos parámetros de entrada: la interfaz bajo la opción -i, y el estado al que se quiere poner con la opción -s. El *script* lee estos dos parámetros y ejecuta el comando *sudo ip link set dev 'INTERFACE' multipath 'STATE'* para cambiar el estado de la interfaz indicada.

Con todo esto, la implementación y configuración estaría acabada, y se tendría MPTCP versión 0 funcionando plenamente en la red virtual creada. En el capítulo 6 se procederá a realizar pruebas de rendimiento y eficiencia cambiando su configuración.

5.3. Desarrollo del Entorno Práctico para MPTCP Versión 1

A continuación, se procede a exponer la implementación del protocolo en su última versión, la versión 1, también en el *kernel* de Linux.

5.3.1. Repaso de Alternativas

Como se ha ido reiterando en las secciones 2.3, 3.3 y 4.2.3, MPTCP versión 1 está soportado por defecto en todas las versiones del *kernel* de Linux desde la v5.6. Pues bien, hay dos posibles alternativas a la hora de implementar el protocolo en alguna de ellas:

- ***Kernel* Oficial de Linux**: si se desea utilizar alguna versión mayor o igual a la v5.6 del *kernel* de Linux, ésta se puede encontrar en los repositorios oficiales. Como se ha mencionado varias veces, estos *kernels* ofrecen soporte al protocolo en su versión 1, por defecto. Es por esto

que es posible obtener una implementación de MPTCP versión 1 en estos *kernels*, pero limitada, ya que en cuanto al *scheduling*, se tiene solo un tipo de programador de paquetes, el *scheduler* por defecto; éste envía los mismos datos por todas las rutas posibles buscando el máximo rendimiento. Así, se pueden hacer pruebas MPTCP, pero con un margen acotado de funcionalidades.

- **Proyecto Linux MPTCP *Upstream***: este proyecto tiene su propio hueco en GitHub [30]. En él se puede encontrar una *wiki* con información acerca de los pasos a seguir para poder usar MPTCP (qué cosas se deben instalar y configurar, qué se debe comprobar, etc). También se adjuntan enlaces a artículos que hablan sobre el protocolo, en el apartado de *ChangeLog* se puede ver la evolución de los diferentes *kernels upstream* y las funcionalidades nuevas de MPTCP que va incorporando cada uno, y por último, se hace un breve resumen sobre las diferencias entre ambos proyectos MPTCP (*out-of-tree vs. upstream*). Además de esta *wiki*, en la parte de código del repositorio [77], se pueden encontrar las diferentes versiones del *kernel* de Linux (a partir de la v5.6), que como se ha mencionó en la sección 3.3, están ampliadas para ir incorporando más funcionalidades del protocolo.

  A fecha 15 de mayo de 2023, el último *kernel* modificado que se encuentra disponible, y por ello el que se va a usar en este trabajo, es el v6.4-rc5 (en realidad se denomina v6.4-rc5+, ya que tiene la base del v6.4-rc5 pero ampliada). Éste no solo incluye soporte al protocolo, además de más características, sino también la implementación de diferentes tipos de *schedulers* mediante módulos eBPF cargados en el *kernel* (esto se comentó en la sección 3.3.1).

En cuanto a la implementación del *path management*, tanto una solución como otra, disponen de dos alternativas:

- ***In-kernel path manager***: este gestor de rutas se implementa en el propio *kernel*. Es configurable a través de un *socket netlink* genérico, y se suele emplear el comando *ip mptcp* (incluido en iproute2-ss200602) para interactuar con él, estableciendo cómo se van a intentar crear los subflujos, qué direcciones van a ser anunciadas, etc.

- ***User-space path manager***: la gestión de rutas en el espacio de usuario trae consigo ventajas como menos código en el *kernel*, limitando posibles errores en él. También tiene desventajas, como los cuellos de botella que se pueden crear por una alta carga en la interacción *kernel*-espacio de usuario sobre el *netlink* genérico; así como la necesidad de más recursos.
  Este gestor de rutas se controla a través del demonio *mptcpd*. Este demonio realiza operaciones relacionadas con la gestión de rutas MPTCP

en el espacio de usuario, interactuando con el *kernel* de Linux en busca de información de conexión, como direcciones IP o interfaces de red disponibles, haciendo uso de una conexión *netlink* genérica. Gracias a esta información, es capaz de solicitar nuevos subflujos, gestionar las solicitudes por parte de los otros extremos, etc.
Esta herramienta se comentó ligeramente en la sección 3.3.1, y tiene su propio repositorio de GitHub [87], donde se pone a disposición del usuario tanto el código fuente, como una guía de las dependencias que son necesarias instalar y los pasos a seguir para tenerlo funcionando.

Además, en ambas soluciones se hace uso de otras herramientas como *mptcpize*, la cual se usa para forzar a una aplicación concreta a que use *sockets* MPTCP, en vez de TCP, en un momento determinado. Esta herramienta se comentó en la sección 3.3.1, y será muy útil a la hora de hacer pruebas en el capítulo 6.

Una vez repasadas ambas soluciones, como cabe esperar, la que se va a escoger para poder ser estudiada es la implementación del proyecto *Upstream*, ya que es la más completa. Aun así, se proporciona un repositorio creado por la estudiante [88], donde se puede consultar la implementación de MPTCP versión 1 en un *kernel* sin modificar (también la versión más nueva de éste, la v6.4-rc5), con el fin de mostrar más información al respecto.

5.3.2. Diseño del Escenario

Para la creación del escenario virtual se va a seguir el mismo modelo que en la figura 5.2, con dos pequeñas variaciones. La figura 5.8 lo muestra:

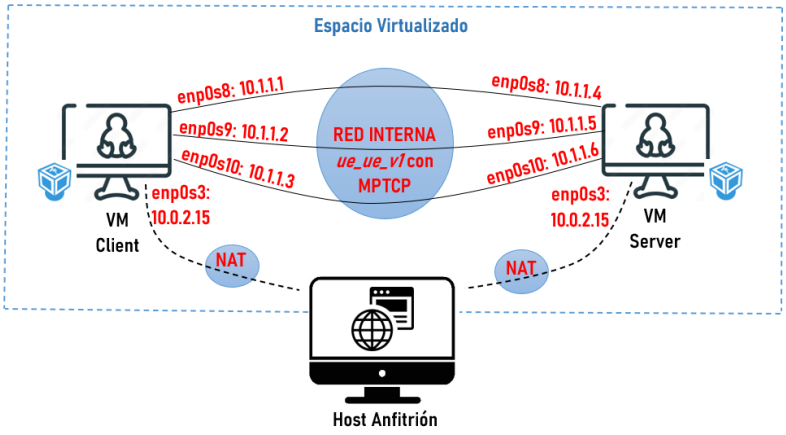


Figura 5.8: Escenario Virtual para MPTCP Versión 1.

Al igual que en la implementación de MPTCP versión 0, ambas máquinas (*Client* y *Server*) tienen 3 interfaces conectadas a la misma red interna, pero

en este caso ésta recibe el nombre de *ue_ue_v1*. Además, ambas máquinas disponen de una cuarta interfaz para su conexión con Internet, al igual que en la implementación anterior.

5.3.3. Creación del Escenario para Albergar MPTCP Versión 1 en la Red

Para la creación del escenario visto en la figura 5.8, se va a necesitar, al igual que para la implementación de MPTCP versión 0, tener instalado el software *Oracle VM VirtualBox*. Dentro de él, se van a crear dos máquinas virtuales *Ubuntu* 22.04 LTS. La versión de *Ubuntu* más moderna es la 22.10, pero al no ser una versión LTS y asegurar el soporte a largo plazo, se escoge la segunda más moderna que sí que lo es, la 22.04. Dicho esto, se crean las máquinas (*Client* y *Server*) siguiendo los pasos de instalación que se van mostrando por pantalla (ahora la instalación se hace manual y no se emplea *Vagrant*).

Una vez se tienen ambas máquinas creadas, y antes de intentar compilar el *kernel upstream* (bajo el nombre de 6.4-rc5+) alojado en GitHub, es necesario instalar una serie de dependencias en cada una, tanto las habituales que se necesitan para cualquier *kernel* [89], como las específicas necesarias para el *kernel* ampliado que se va a compilar e instalar [90].
Es muy importante realizar correctamente la instalación de todas las herramientas que se muestran en los dos enlaces proporcionados, de lo contrario, la compilación del *kernel* dará problemas.

Así pues, con ambas máquinas creadas y todas las dependencias necesarias instaladas, se puede proceder a compilar el *kernel*. Para ello, se accede al repositorio de GitHub que alberga el proyecto [77], y se descarga el código fuente del *kernel* en cada una. Una vez descargado y descomprimido, dentro de la carpeta se ejecutan los siguientes comandos:

```
1. sudo make clean
2. sudo make menuconfig
```

Cabe destacar que al llevar a cabo el comando *sudo make menuconfig*, se abre un menú de configuración en el que se pueden habilitar o deshabilitar opciones, con el fin de customizar a nuestro gusto el *kernel*. Es importante comprobar dentro de él que la sección *Networking support->Networking options->TCP/IP networking->MPTCP protocol (MPTCP)* está marcada con un *. Además, se debe modificar el archivo *.config* y borrar la comprobación de certificados Debian para X509 que se encuentra en las secciones *CONFIG_SYSTEM_TRUSTED_KEYS* y *CONFIG_SYSTEM_REVOCATION_KEYS*. Estos certificados no son necesarios, y si no se borran la compilación da error.
Una vez hecho esto, se ejecuta:

```
3. sudo make -j‘nproc‘
4. sudo make -j‘nproc‘ bindeb-pkg
```

Estos comandos realizan la compilación del *kernel* y generan cuatro paquetes: linux-headers-*.deb, linux-image-*.deb, linux-libc-*.deb y linux-image-dbg*.deb. El último está relacionado con temas de *debug*, por lo que no es necesario instalarlo y se puede eliminar. Los demás paquetes se deben instalar con el comando:

```
5. sudo dpkg -i *.deb
```

Una vez instalados, se podría hacer un reinicio de ambas máquinas y ya se tendría el nuevo *kernel* instalado; sin embargo, si por el momento se quiere utilizar siempre el *kernel* 6.4-rc5+, pero se poseen varias versiones en la máquina y no se quiere borrar ninguna, se puede hacer lo siguiente:

Se descarga el archivo *grub-menu.sh* del repositorio [91], y se ejecuta. Éste muestra un menú en el que se visualiza una secuencia de números al lado de cada uno de los *kernels* disponibles en el sistema. La secuencia que pertenece al *kernel* 6.4-rc5+, que será de la forma '1>0', se debe cambiar por el valor que tenga GRUB_DEFAULT en el archivo */etc/default/grub* (de la forma GRUB_DEFAULT="1>0"). Seguidamente, se actualiza grub mediante *sudo update-grub*, y se reinicia la máquina. De esta manera, se garantiza que cada vez que la máquina se inicie de nuevo, el *kernel* que se cargará por defecto será el 6.4-rc5+. Cuando se quiera usar otra versión, basta con modificar el archivo */etc/default/grub* de nuevo.

Así, se tienen dos máquinas con dicho *kernel* compilado e instalado sin problemas.

Configuración del Sistema

Lo siguiente que se debe hacer es configurar tanto las direcciones IP de las interfaces de cada máquina, como el *routing*, además del *scheduling* y *path management*.

Como se observa en la figura 5.8, cada máquina tiene 3 interfaces conectadas a una misma red interna *ue_ue_v1*, a través de las cuales se implementará MPTCP. Además, se tiene una cuarta interfaz que se usará para conectarse a Internet, por lo que es una interfaz NAT; a ésta se le asigna IP mediante *Dynamic Host Configuration Protocol* (DHCP), por lo que no hay que configurarla. Así pues, lo primero que hay que hacer es darle dirección IP a cada una de las interfaces de la red interna, para ello se accede al archivo */etc/netplan/01-network.manager-all.yaml* y se configura:

- Para la máquina *Client*: enp0s8 (10.1.1.1), enp0s9 (10.1.1.2) y enp0s10 (10.1.1.3).

- Para la máquina *Server*: enp0s8 (10.1.1.4), enp0s9 (10.1.1.5) y enp0s10 (10.1.1.6).

Se debe ejecutar el siguiente comando para que los cambios se guarden:

```
sudo netplan apply
```

Una vez se tienen todas las interfaces con su IP asignada, se debe llevar a cabo el *routing*. Para ello, se ejecutan dos *scripts* de *shell* que siguen los pasos citados en [85]. Estos *scripts* están disponibles en el Apéndice B.
Una vez ejecutados, al igual que en la sección 5.2.3, se puede comprobar la correcta configuración de direcciones mediante una serie de comandos mostrados en la figura 5.9:

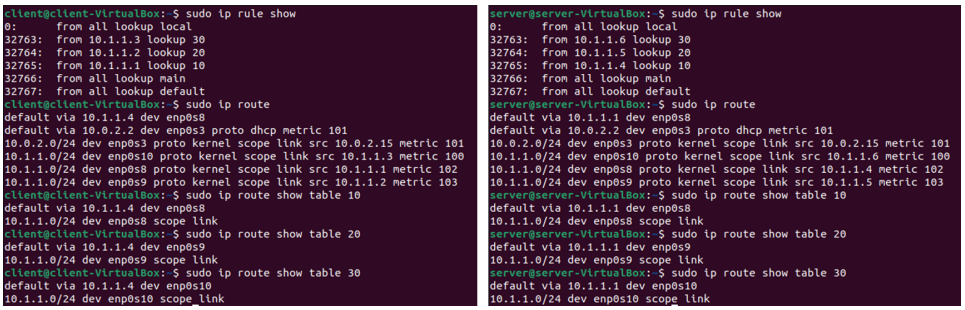


Figura 5.9: Configuración del *Routing* en Ambas Máquinas Virtuales (MPTCP versión 1).

Cabe destacar que la interfaz con mayor prioridad es la enp0s10 en ambas máquinas, esto implica que el cliente la utilizará para conectarse inicialmente al servidor. En el caso de éste, como el cliente se conecta a la IP 10.1.1.4 explícitamente, no se utilizará ésta para la conexión inicial.

Lo siguiente que se debe configurar es el *path manager*. Como se vio en la sección 5.3.1, hay dos tipos de gestor de rutas, en este caso se va a escoger el *in-kernel path manager*, debido a que su configuración es más simple de mostrar. Cabe destacar, que el *kernel* instalado trae por defecto el demonio *mptcpd*, por lo que será necesario pararlo antes de hacer algún tipo de configuración de rutas (en caso de que se quiera ver como funciona éste último, se puede obtener información en [88]).

Volviendo al *in-kernel path manager*, éste se configura con el comando *ip mptcp* y, a diferencia del *path manager* en MPTCP versión 0, en este caso el usuario es el que decide cómo se van a ir gestionando las direcciones. Para ello, es necesario establecer de qué tipo van a ser las interfaces de cada máquina, lo que se va a llamar *endpoint*. Estos pueden ser de 4 tipos:

- ***Subflow***: las interfaces configuradas de esta forma enviarán mensajes MP_JOIN a otras interfaces, con el fin de iniciar nuevos subflujos.

- ***Signal***: una interfaz configurada así, permite que su IP sea anunciada mediante mensajes ADD_ADDR. Ella nunca inicia un subflujo (no manda mensajes MP_JOIN).
- ***Fullmesh***: las interfaces configuradas de esta forma enviarán mensajes MP_JOIN para iniciar nuevos subflujos a cada una de las demás interfaces de la otra máquina, formando una topología de subflujos todas-a-todas.
- ***Backup***: la interfaz que tenga esta opción hace que los subflujos creados utilizándola se hagan mediante mensajes MP_JOIN con el *flag backup* a 1, por lo que no se utilizarán hasta que sea necesario. Una interfaz no puede ser *backup* y *signal* a la vez, debido a que el mensaje ADD_ADDR se lee antes del procesamiento del *endpoint*, y por ende, la respuesta a este mensaje es un MP_JOIN con el *flag backup* a 0 antes de que se lea que la interfaz se había puesto con *backup* a 1.

Para reproducir el mismo escenario que se mostró en la primera prueba de verificación de MPTCP versión 0, pero simulando un caso real con el cliente haciendo peticiones de creación de conexión, y el servidor a la escucha, será necesario configurar todas las interfaces del cliente como *subflow fullmesh*, y las del servidor como *signal*.

Además, se debe establecer el número máximo de subflujos adicionales que se van a poder crear en la conexión MPTCP entre las dos máquinas (sin contar con el que se crea para hacer la conexión), y el número máximo de mensajes ADD_ADDR que se van a permitir. Estas dos últimas configuraciones tienen como objetivo intentar evitar los ataques que se comentaron en la sección 3.2.4, los cuales motivaron el surgimiento de esta nueva versión del protocolo.

Dichas configuraciones se realizan mediante los *scripts* de las figuras 5.10 y 5.11, y como se puede ver, antes de todo ello es importante parar el demonio *mptcpd* y eliminar los *endpoints* que se hayan creado al iniciar la máquina.

```
#!/bin/bash

#Deshabilitar mptcpd y borrar endpoints anteriores
sudo systemctl stop mptcp.service
sudo ip mptcp endpoint flush

#Configuracion de endpoints
sudo ip mptcp endpoint add 10.1.1.1 dev enp0s8 subflow fullmesh
sudo ip mptcp endpoint add 10.1.1.2 dev enp0s9 subflow fullmesh
sudo ip mptcp endpoint add 10.1.1.3 dev enp0s10 subflow fullmesh

#Establecimiento de limites MPTCP
sudo ip mptcp limits set subflow 8 add_addr_accepted 8
```

Figura 5.10: Configuración del *Path Manager* en el Cliente.

```
#!/bin/bash

#Deshabilitar mptcpd y borrar endpoints anteriores
sudo systemctl stop mptcp.service
sudo ip mptcp endpoint flush

#Configuracion de endpoints
sudo ip mptcp endpoint add 10.1.1.4 dev enp0s8 signal
sudo ip mptcp endpoint add 10.1.1.5 dev enp0s9 signal
sudo ip mptcp endpoint add 10.1.1.6 dev enp0s10 signal

#Establecimiento de limites MPTCP
sudo ip mptcp limits set subflow 8 add_addr_accepted 0
```

Figura 5.11: Configuración del *Path Manager* en el Servidor.

Como se observa en las figuras 5.10 y 5.11, el número máximo de subflujos adicionales se establece a 8 en ambas máquinas, para que pueda haber subflujos entre las 3 interfaces de una con las 3 de la otra (además este es el valor máximo que se puede establecer). Por otro lado, en el cliente se van a aceptar también hasta 8 anuncios de direcciones, mientras que en el servidor no hace falta establecer límite debido a que él solo recibe mensajes MP_JOIN.
De esta forma, el *path manager* en ambas máquinas queda configurado. Se puede comprobar dicha configuración con los comandos que se muestran en la figura 5.12:

```
client@client-VirtualBox:~$ sudo ip mptcp endpoint
10.1.1.1 id 1 subflow fullmesh dev enp0s8
10.1.1.2 id 2 subflow fullmesh dev enp0s9
10.1.1.3 id 3 subflow fullmesh dev enp0s10
client@client-VirtualBox:~$ sudo ip mptcp limits show
add_addr_accepted 8 subflows 8
```

```
server@server-VirtualBox:~$ sudo ip mptcp endpoint
10.1.1.4 id 1 signal dev enp0s8
10.1.1.5 id 2 signal dev enp0s9
10.1.1.6 id 3 signal dev enp0s10
server@server-VirtualBox:~$ sudo ip mptcp limits show
add_addr_accepted 0 subflows 8
```

Figura 5.12: Configuración del *Path Manager* en Ambas Máquinas Virtuales (MPTCP versión 1).

Por último, queda por configurar el *scheduling*. Como se ha mencionado al principio de la sección 5.3.1, el *kernel upstream* escogido cuenta, además de con el *scheduler* por defecto integrado en él, igual que pasaba en el *kernel* oficial, con una serie de *schedulers* que se cargan como módulos eBPF. Los distintos *schedulers* entre los que se puede elegir son:

- ***Default***: este programador de paquetes no funciona igual que el *default* en MPTCP versión 0, sino igual que el *redundant*. Envía todos los datos por todos los subflujos disponibles con el fin de alcanzar el máximo rendimiento, repitiendo el comportamiento del *scheduler redundant*.

- ***Round-Robin*** : este *scheduler* siempre elige el siguiente subflujo disponible para enviar datos de forma *round-robin*. Si no hay siguiente subflujo disponible, elige el primero.

- ***Redundant*** : envía los mismos datos por todos los subflujos de forma redundante.

- ***First*** : se envían los datos por el primer subflujo que haya disponible, que normalmente suele ser el que se utiliza para hacer la conexión inicial cliente-servidor.

- ***Backup*** : este último *scheduler* escoge el primer subflujo que no sea de *backup* para enviar los datos, por lo que suele coger el subflujo que se forma para hacer la conexión inicial, funcionando igual que el *first*.

Como se ha explicado, el *scheduler* por defecto (*default*) ya viene integrado en el propio *kernel*, mientras que los demás se deben cargar en él. Para llevar a cabo esto, se siguen los siguientes pasos:

1. Lo primero que se debe hacer es acceder, dentro de cada máquina virtual, a la carpeta que contiene el código fuente del *kernel*, en concreto, al directorio *mptcp_net-next/tools/testing/selftest/bpf/progs/*.

2. Dentro de él, se pueden encontrar 4 ficheros *.c* bajo los nombres *mptcp_bpf_red*, *mptcp_bpf_rr*, *mptcp_bpf_first* y *mptcp_bpf_bkup*. Como cualquier fichero en lenguaje C, es necesario compilarlo para poder crear el archivo objeto (*.o*), el cual sea el que se ejecute, o en este caso, se cargue al *kernel*.

3. Para compilar cualquiera de los 4 archivos, se hace uso del siguiente comando:

```
sudo clang -O2 -target -bpf -g -c mptcp_bpf_X.c -o mptcp_bpf_X.o
```

Como se observa, se utiliza la herramienta *clang*, un compilador de código abierto utilizado principalmente para el lenguaje de programación C y C++. A ella se le suman una serie de argumentos:

- **-O2**: para aplicar un nivel moderado de optimizaciones al código.
- **-target -bpf**: *target* se emplea para indicar el destino de la compilación, al ir detrás bpf, se está indicando que el objetivo es el entorno *Berkeley Packet Filter* (BPF) (éste es un mecanismo que permite ejecutar código personalizado dentro del *kernel*, en lo referente a filtrado y procesamiento de paquetes de red).
- **-g**: con esta opción se indica que se desea añadir información de depuración en el archivo objeto. Es opcional.
- **-c mptcp_bpf_X.c**: se le indica al compilador que genere un archivo objeto a partir del archivo fuente, el cual será *mptcp_bpf_red.c*, *mptcp_bpf_rr.c*, *mptcp_bpf_first.c* o *mptcp_bpf_bkup.c*.
- **-o mptcp_bpf_X.o**: se especifica el nombre del archivo objeto que se va a obtener (*mptcp_bpf_red.o*, *mptcp_bpf_rr.o*, *mptcp_bpf_first.o* y *mptcp_bpf_bkup.o*).

4. Una vez están todos los *schedulers* compilados, se deben cargar en el *kernel*. Para ello, se utiliza un nuevo comando:

```
sudo bpftool struct_ops register mptcp_bpf_X.o
```

Se emplea la herramienta *bpftool*. Esta se encarga de la inspección y manipulación de programas y mapas eBPF (lectura y escritura, compilación, búsqueda de errores, etc). Se le añaden dos subcomandos, además del archivo objeto a cargar:

- ***struct_ops***: este subcomando permite registrar una serie de funciones que se utilizan para operar con objetos BPF, lo que recibe el nombre de estructura de operaciones o *struct_ops*.
- ***register***: se registra la estructura de operaciones incluida en el archivo objeto que se indica tras él.

Siguiendo estos pasos, los diferentes *schedulers* ya están compilados y cargados en el *kernel*. Para comprobarlo se puede ejecutar el siguiente comando, el cual proporciona una salida como la de la figura 5.13:

```
sudo bpftool struct_ops show
```

```
client@client-VirtualBox:~$ sudo bpftool struct_ops show
4: rr              mptcp_sched_ops
5: red             mptcp_sched_ops
6: first           mptcp_sched_ops
7: bkup            mptcp_sched_ops
```

Figura 5.13: Comprobación de la Compilación y Carga de los *Schedulers* en el *Kernel*.

Una vez se tienen todos los tipos de *scheduler* preparados (el *default* ya viene soportado, y los demás están compilados y cargados en el *kernel*), se puede seleccionar el que se desee utilizar mediante el uso de variables *sysctl*, al igual que se hacía en la implementación de MPTCP versión 0.
Dentro de los archivos *.c* de cada *scheduler*, hay una variable denominada *.name* a la que se le asigna el nombre que recibe dicho *scheduler*; pues bien, este nombre es el que hay que asignarle a la variable *sysctl* relacionada con los *schedulers*:

```
sudo sysctl net.mptcp.scheduler =  [default/ bpf_red/ bpf_rr/
bpf_first/ bpf_bkup]
```

Si se requiere información sobre las diferentes variables *mptcp sysfs* que se tienen, se puede echar un vistazo a [92], o ejecutar el comando:

```
sudo sysctl net.mptcp
```

Su salida se observa en la figura 5.14. Se puede ver como hay varias variables:

- ***net.mptcp.add_addr_timeout*=120**: establece a 120 segundos el tiempo en el que se agotará un mensaje ADD_ADDR reenviado que no ha sido reconocido.
- ***net.mptcp.allow_join_initial_addr_port*=1**: si se iguala a 1 se permite que los pares envíen solicitudes de creación de subflujo a la IP y puerto utilizados por el subflujo inicial.
- ***net.mptcp.checksum_enabled*=0**: se puede habilitar la suma de comprobación DSS si es valor es distinto de 0.
- ***net.mptcp.enabled*=1**: controla si se pueden crear *sockets* MPTCP. Si está a 1 es que sí.
- ***net.mptcp.pm_type*=0**: se establece el gestor de rutas en el *kernel*. Si es igual 1, se indica que se va a utilizar el *path manager* en el espacio de usuario, el demonio *mptcpd*. Si, por el contrario, se pone a 0, se utilizará el *in-kernel path manager*.
- ***net.mptcp.scheduler=default***: establece el tipo de *scheduler*.

- ***net.mptcp.stale_loss_cnt*=4**: se establece el número de intervalos de retransmisión a nivel MPTCP sobre un subflujo determinado necesarios para declararlo obsoleto. Un valor bajo permite declarar un subflujo como reserva más rápidamente, mientras que un valor alto permite escenarios de máxima utilización de los enlaces y con más pérdidas.

```
client@client-VirtualBox:~$ sudo sysctl net.mptcp
net.mptcp.add_addr_timeout = 120
net.mptcp.allow_join_initial_addr_port = 1
net.mptcp.checksum_enabled = 0
net.mptcp.enabled = 1
net.mptcp.pm_type = 0
net.mptcp.scheduler = default
net.mptcp.stale_loss_cnt = 4
```

Figura 5.14: Variables *sysctl* en MPTCP Versión 1.

En cuanto al control de congestión, no hay un mecanismo concreto implementado, pero se está estudiando la posibilidad de introducirlo en un futuro mediante BPF, al igual que los *schedulers*.

Verificación del Sistema

Al igual que con MPTCP versión 0, las pruebas de rendimiento se realizarán en el capítulo 6, pero para verificar que simplemente el protocolo funciona, seguidamente se expone una pequeña prueba.

Se escoge una configuración simple: *scheduler* por defecto, y la misma configuración de *path manager* mostrada en las figuras 5.10 y 5.11. Para realizar las pruebas se hace uso de dos comandos nuevos:

Server

```
mptcpize run iperf3 -s & ifstat
```

Client

```
mptcpize run iperf3 -c 10.1.1.4 -t 300 & ifstat
```

Se utiliza la herramienta comentada en la sección 5.3.1, *mptcpize*, la cual fuerza, mediante la sentencia *mptcpize run*, la creación de *sockets* MPTCP en lugar de TCP para una determinada aplicación, en este caso, *iperf3*. *iperf3* es una herramienta para realizar pruebas de rendimiento, es la versión más nueva de la herramienta *iperf*, utilizada también para las pruebas con MPTCP versión 0.
Como argumentos se añaden:

- **-s**: para poner un servidor de escucha en el servidor.
- **-c <IP>**: indica la dirección IP del servidor a la que se va a conectar el cliente.
- **-t <tiempo>**: indica el tiempo que va a durar la prueba.

Además, *iperf3* se ejecuta junto con otra herramienta, *ifstat*, ya comentada en MPTCP versión 0.

Si se realiza una prueba ejecutando dichos comandos en cliente y servidor, se obtiene lo observado en la figura 5.15. Como se puede ver, se envían datos por las 3 interfaces de cada máquina. La prueba de que se envían los mismos datos por todos los subflujos creados, así como la creación de subflujos entre cada una de las interfaces disponibles, se lleva a cabo en el capítulo 6, en el cual se evalúa el funcionamiento y rendimiento de cada tipo de *scheduler*.

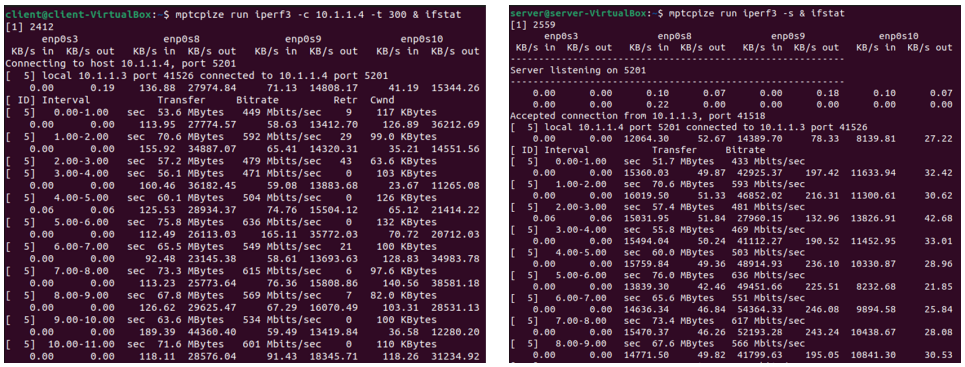


Figura 5.15: Prueba Inicial de MPTCP Versión 1.

Además, al igual que en la versión 0 del protocolo, las interfaces se pueden establecer como *backup*, siempre y cuando se utilice el *default scheduler*. Las interfaces del cliente, que son las únicas que se pueden poner como *backup*, como ya se vio en la explicación de *ip mptcp* en esta sección, deben estar configuradas únicamente como *subflow* (en los demás casos este concepto no está bien configurado todavía). Para poner una interfaz como *backup*, se puede hacer desde un inicio mediante los comandos mostrados en las figuras 5.10 y 5.11 (mensaje MP_JOIN con el *flag backup* a 1), o en mitad de la comunicación (mensaje MP_PRIO). Para hacerlo de esta última forma, se utiliza un nuevo comando:

```
sudo ip mptcp endpoint change <IP> backup
```

La opción *change* de *ip mptcp* está disponible en la última versión de *iproute2*, la cual no viene instalada por defecto y se puede descargar del repositorio [93]. Al instalarla, también se instala una nueva versión de *ifstat*, si se quiere

volver a la anterior, se puede encontrar en [94].

A continuación, se pone la interfaz *enp0s9* del cliente como *backup* (se pone después de iniciar la comunicación, pero también se puede haber configurado antes), y se hace una prueba de funcionamiento. El resultado se muestra en la figura 5.16:

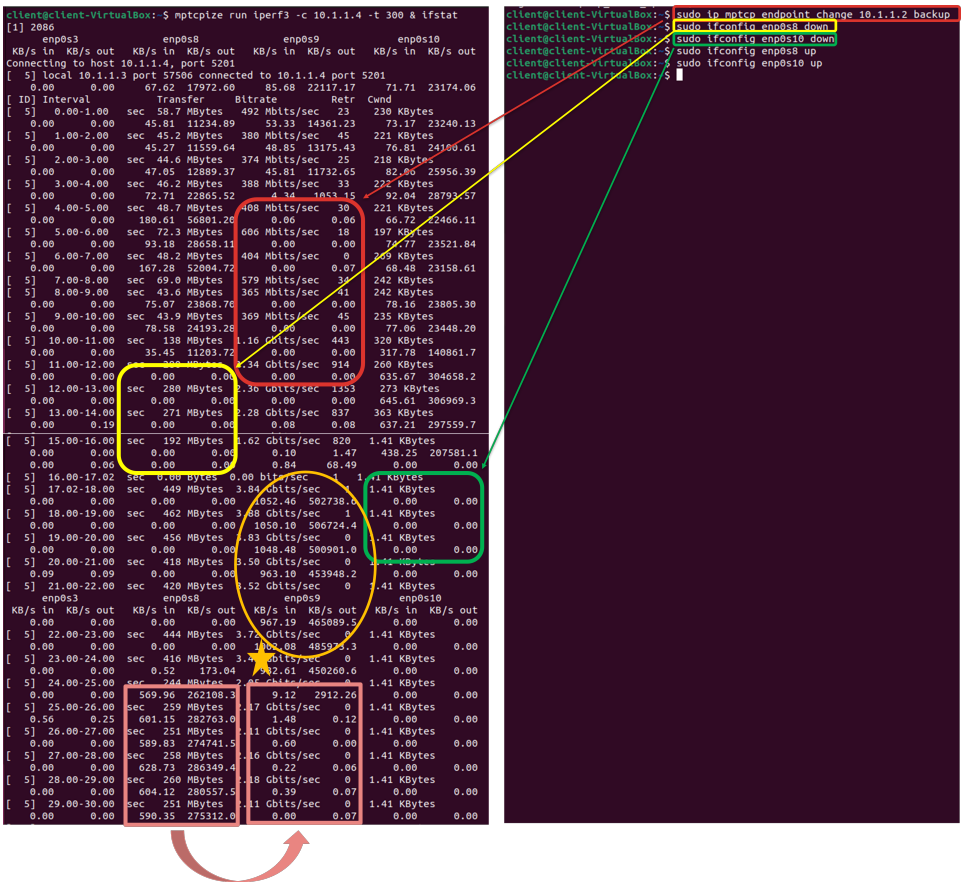


Figura 5.16: Subflujos de Reserva en una Conexión con MPTCP Versión 1.

Observando la figura 5.16, se puede ver cómo cuando *enp0s9* se pone en *backup*, se deja de utilizar (color rojo). Seguidamente, se echa la interfaz *enp0s8* abajo (color amarillo), y se observa como se sigue utilizando la *enp0s10*. En el momento en el que ésta se cae (color verde), la interfaz *enp0s9* se reactiva salvando la conexión (color naranja con estrella).
Después de esto, cuando *enp0s8* vuelve a funcionar, *enp0s9* se pone en *standby* como debe (color rosa). Sin embargo, al reactivar la segunda interfaz caída, *enp0s10*, el comportamiento de *enp0s9* no es el que debe, ya que se reactiva también (esto último no se muestra en la figura, ya que este suceso ocurre mucho tiempo después, al tardar ambas interfaces en reaccionar). Al

estar el proyecto en pleno desarrollo, la implementación del concepto *backup*, con mucha seguridad, no se haya perfeccionado del todo todavía.

Con esto queda implementado y configurado MPTCP versión 1 en esta segunda red virtual.

Una vez se han expuesto los pasos que se han seguido en la creación y configuración de los escenarios virtuales que albergan MPTCP versión 0 y MPTCP versión 1, se concluye este capítulo. En el próximo capítulo se realizarán todas las pruebas y verificaciones detalladas que corroboren el buen funcionamiento de ambos sistemas.

Capítulo 6

Pruebas y Experimentación

En este capítulo se van a abordar las pruebas realizadas para la comprobación del correcto funcionamiento de MPTCP en sus versiones 0 y 1, en los sistemas virtuales creados. Pero antes de ello, se analizará una captura de *Wireshark* para mostrar los mensajes que se intercambian ambos extremos, y corroborar que cumplen con la señalización explicada en el capítulo 3.

Para realizar las pruebas se va a hacer uso de varias herramientas de línea de comandos, algunas ya vistas como *mptcpize* o *iperf3*, y otras nuevas como *tc*. Los resultados obtenidos se mostrarán haciendo uso de *Grafana* y *MatLab*.

Mediante *Grafana* se obtendrán gráficas que muestren el rendimiento en términos de *throughput* de MPTCP versión 0, manteniendo fijo un tipo de *path manager*, y valorando cómo trabaja cada uno de los *schedulers* disponibles. Además, para esta versión del protocolo se montará un escenario real y se hará una prueba con 5G y Wi-Fi. Por otro lado, también se valorará el *throughput* con *Grafana* para la versión 1, al igual que con versión 0. Para todo ello, en los escenarios virtuales se limitará la velocidad (en términos de tasa) de todas las interfaces de cada máquina a 100 Mb/s mediante la herramienta *tc*, con el fin de observar mejor cómo se comporta el protocolo. Esto se ha hecho así debido a que las interfaces tienen una velocidad de 1000 Mb/s. Sin embargo, ésta nunca se alcanza y hay mucha variabilidad en los resultados debido a las limitaciones que tiene el portátil usado, por lo que al hacer pruebas, no se puede ver bien qué cantidad de datos se transmiten en cada una. Limitando todas a 100 Mb/s, es más fácil observar cuántos datos se están enviando y recibiendo en cada interfaz, y por ende, en los subflujos. Además se mostrará a modo complementario, usando *MatLab*, una comparación en ambas versiones de las *Cumulative Distribution Function* (CDF) de cada uno de sus *schedulers*, con el fin de visualizar la diferencia en cuanto a distribución acumulativa de *throughput* que tiene cada uno.

Para finalizar el capítulo, se hace un breve resumen de las complicaciones

encontradas a lo largo del proyecto, debido a que el tema a estudiar se encuentra actualmente en pleno desarrollo.

6.1. Estudio de la Señalización del Protocolo

Para mostrar, no solo cómo funcionan los escenarios virtuales montados y los rendimientos que se alcanzan, sino los mensajes que se intercambian ambos extremos para que esto suceda así, se ha hecho una prueba como la que se lleva a cabo en la verificación del sistema del apartado 5.3.3. Capturando los mensajes con *Wireshark*, se puede observar cómo el protocolo MPTCP sigue los siguientes pasos:

1. Primero se produce un intercambio de mensajes SYN con la opción MP_CAPABLE entre las interfaces de los extremos que hacen la comunicación (en este caso, 10.1.1.3 del cliente y 10.1.1.4 del servidor). Este protocolo va dentro de las opciones de TCP, y la opción en concreto, dentro del campo *subtype* a valor 0x01. Los mensajes son los marcados en negro en la figura 6.1.

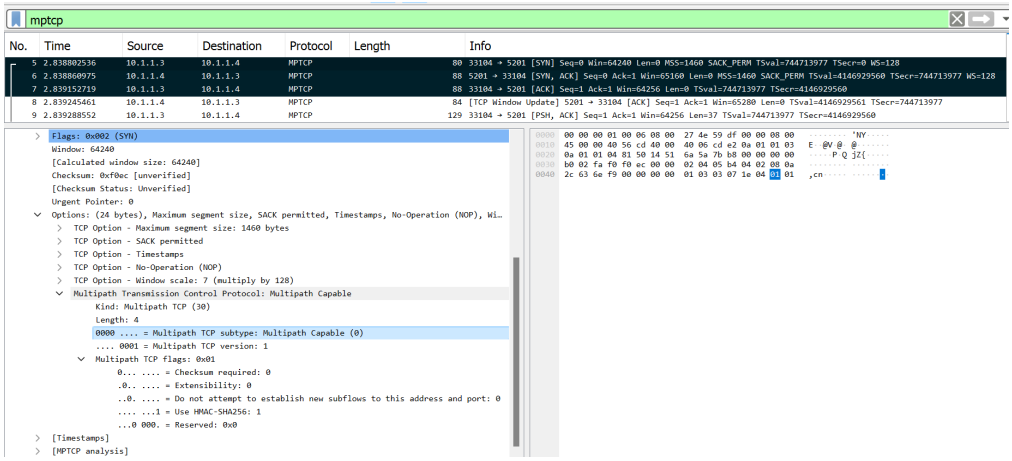


Figura 6.1: Intercambio MP_CAPABLE.

2. Seguidamente, ambas interfaces se empiezan a mandar datos usando el subflujo creado entre ellas. Para ello, se utiliza la opción DSS (los mensajes de datos usando DSS son los que aparecen en blanco en la figura 6.2) vista en el capítulo 3. Además, entre medias se producen anuncios de direcciones (marcados en negro) mediante la opción ADD_ADDR. En concreto, el servidor anuncia al cliente las direcciones IP de las otras dos interfaces que tiene disponibles, haciendo uso de la interfaz con la que hizo la conexión (10.1.1.4 anuncia 10.1.1.5 y 10.1.1.6).

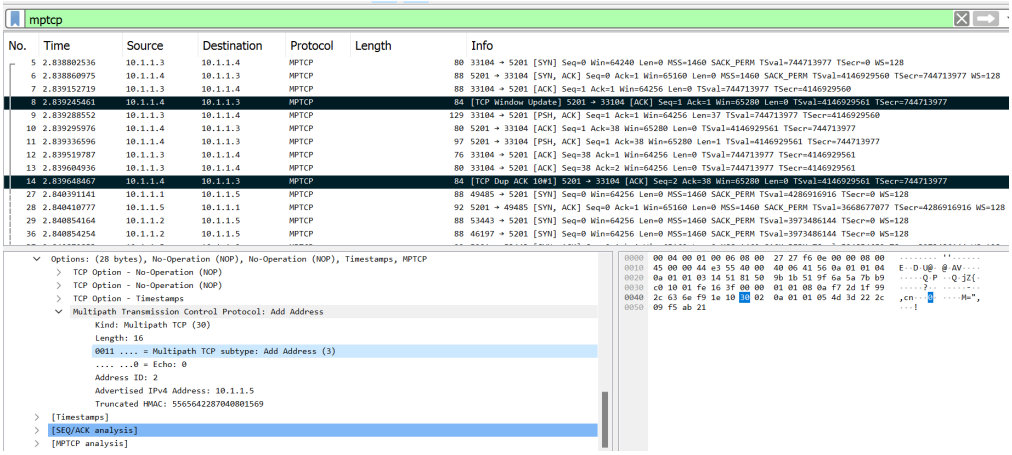


Figura 6.2: Mensajes ADD_ADDR y DSS.

3. Al conocer el cliente todas las interfaces del servidor, empieza a mandar mensajes MP_JOIN a estas, con el fin de crear nuevos subflujos. En la figura 6.3 se ha coloreado en un color diferente cada intercambio SYN+MP_JOIN. Se puede observar cómo se crean subflujos entre las IP 10.1.1.1 y 10.1.1.2 con la 10.1.1.5, la 10.1.1.1. con la 10.1.1.4, etc. De esta forma, se siguen creando subflujos hasta obtener uno por cada par de IP (con un máximo de 8 subflujos en toda la conexión).

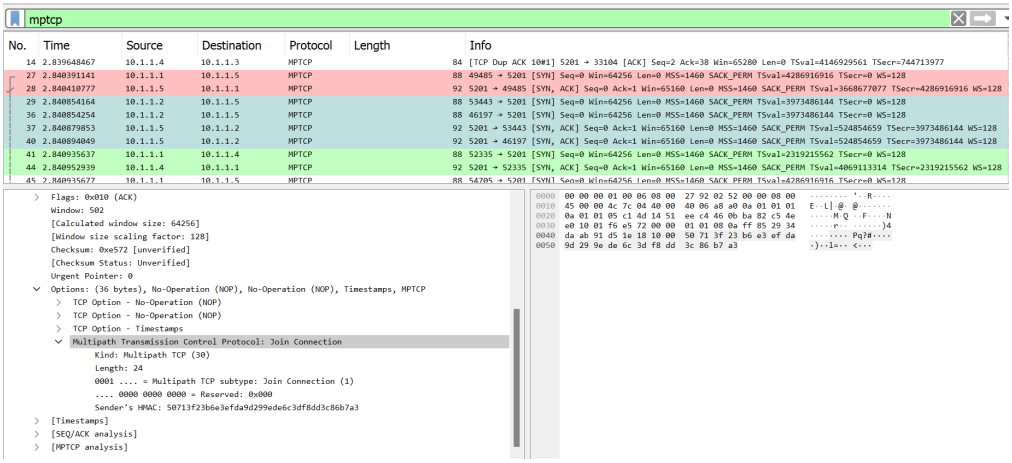


Figura 6.3: Intercambio MP_JOIN.

4. A partir de este momento, cliente y servidor se empiezan a mandar y recibir datos por todos los subflujos creados, hasta que uno de ellos decide cortar la conexión. En dicho momento, en cada subflujo se envían mensajes DATA_FIN, que son reconocidos por mensajes DATA_ACK. Una vez producido este intercambio, se produce uno nuevo, pero esta

vez de mensajes TCP FIN. Éste se puede observar en la figura 6.4, donde con cada color se muestra un intercambio FIN en un subflujo diferente. Cuando esto termina, la conexión queda cerrada.

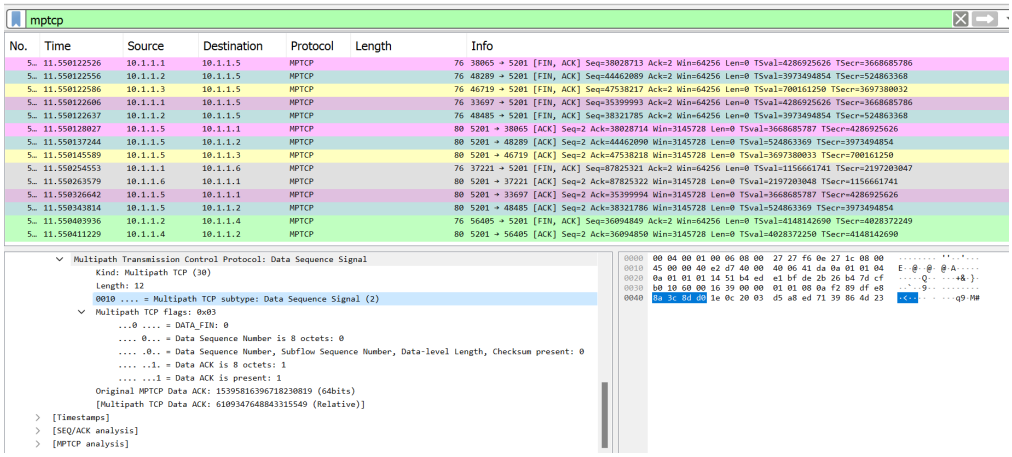


Figura 6.4: Intercambio FIN.

Explicado esto, se puede corroborar como en cuanto a señalización, lo visto en teoría cumple a la perfección con lo que llega a ocurrir en la práctica. Ahora se puede pasar a la experimentación con MPTCP en ambas versiones.

6.2. Montaje y Configuración del Entorno de Pruebas

Antes de realizar ninguna prueba, se debe configurar el entorno de experimentación, lo cual se realiza siguiendo los pasos del repositorio [95]:

1. Una de las máquinas va a mandar las estadísticas obtenidas al software *Grafana*, el cual va a mostrar las gráficas elaboradas a partir de dichos resultados. Pues bien, en este caso va a ser el servidor, *mptcpUe1* en MPTCP versión 0 y *Server* en MPTCP versión 1, los que van a cumplir con este rol.

2. Dentro de las máquinas servidoras, se descarga el repositorio *multi-technology_testbed_v0* (en la versión 0 ya se hizo siguiendo el Apéndice A, y en la versión 1 se debe descargar). Pues bien, dentro de este repositorio se crea un directorio nuevo como *vagrant/stats*.

3. Dentro de dicho directorio se llevan a cabo una serie de tareas:

   a) Descarga e instalación de *Grafana* versión 8.0.1.

b) Descarga de *Prometheus* 2.27.1. Esta base de datos es la que se va a emplear para guardar las estadísticas que se van obteniendo de las pruebas.

c) Instalación de la biblioteca *prometheus_client*, y descarga de un componente de *prometheus* llamado *node exporter*, en su versión 1.1.2. El *node exporter* se ejecuta en la máquina y es el que permite recolectar los datos o métricas, mientras que *prometheus_client* permite exponer las métricas en un formato compatible con *Prometheus*, para que éste pueda leer los datos con facilidad.

d) Descarga del repositorio [95] dentro de la máquina. Este repositorio contiene archivos relacionados con la configuración y el panel de control de *Grafana*.

e) Por último, se detiene el servicio de *Grafana* y se copian los ficheros de configuración de *Prometheus* y de *Grafana* en sus directorios correspondientes. Seguidamente, se reinicia el servicio de nuevo.

f) Los últimos pasos llevan a cabo la configuración de *Grafana* desde la máquina, de forma remota. Sin embargo, esto no se va a hacer así, y se va a configurar en el propio software.

4. Una vez se han seguido los pasos anteriores, se debe acceder a *Grafana* y configurarla. Ésta es accesible en el puerto 3000 de *localhost* dentro de la máquina, ya que es donde se ha instalado. Sin embargo, para poder ver las gráficas de una forma más cómoda en el *host* anfitrión, se accede a la configuración de red de la máquina, y en el adaptador NAT se hace reenvío de puertos, en concreto, del puerto 3000 de la máquina virtual al puerto 13000 del *host* anfitrión. De esta forma, cuando se acceda a *Grafana* en el navegador del portátil, en realidad se estará redirigiendo la petición al puerto 3000 de la máquina virtual.

5. Una vez hecho el reenvío de puertos o *port forwarding*, se puede volver a iniciar la máquina. Dentro de ella, en el repositorio [95] descargado, se encuentra el archivo *start_stats.sh*. Este archivo comienza a mandar las estadísticas a *prometheus*, una vez ejecutado (aunque los datos vayan a ser casi nulos ya que no se está haciendo ninguna prueba), se puede acceder a *https://localhost:13000* en el *host* anfitrión y observar cómo aparece la página de inicio de *Grafana*, ya que se está haciendo correctamente el reenvío de puertos.

6. Dentro de *Grafana* se debe acceder a *Configuration/Data Sources* y añadir una base de datos *Prometheus* con la IP de la máquina virtual para NAT y puerto 9090 (9090 es el puerto por defecto en el que se encuentra disponible *Prometheus*).
   Una vez añadida la base de datos, es necesario crear un panel en el que

se visualizarán posteriormente las gráficas. Esto se realiza accediendo al símbolo + en el menú de inicio de *Grafana*, e importando el modelo *json* que configurará el panel requerido. Para este proyecto se va a usar el archivo *mptcp_dashboard_UGR_nucs.json*, que se puede encontrar en [95], por lo que también es importante tener este repositorio descargado en el *host* anfitrión, no solo en las máquinas virtuales. Las únicas modificaciones que se realizan en el archivo son: cambiar el nombre de las interfaces según se hayan establecido en la máquina, y determinar el nombre que van a tener cada una de las gráficas obtenidas.

Siguiendo los pasos que se han ido describiendo, ya se tiene el entorno de pruebas montado y preparado para ser probado.

6.3. *Throughput* en MPTCP Versión 0

Como se ha mencionado en la introducción de este capítulo, la intención es probar el escenario virtual en el que se ha implementado MPTCP versión 0. Para ello, siguiendo los pasos mencionados en la sección 6.2, se crea un panel o *dashboard* en *Grafana* que muestre 4 gráficas: tráfico recibido del cliente, tráfico enviado al cliente, tráfico acumulado recibido del cliente y tráfico acumulado enviado al cliente, en las 3 interfaces o caminos disponibles, *eth1* o *path* 1, *eth2* o *path* 2 y *eth3* o *path* 3. El archivo *json* utilizado también muestra estadísticas de retardo, pero estas no se van a tener en cuenta ya que no supone una métrica tan relevante a estudiar en este caso.

Lo que se quiere probar es la funcionalidad de cada uno de los *schedulers*, por lo tanto, se configuran ambas máquinas con el *path manager fullmesh*, y se va cambiando el tipo de *scheduler* (para ello se usan los comandos vistos en la sección 5.3.3).

A la hora de hacer las pruebas, se limita la velocidad (en términos de tasa) de las interfaces de ambas máquinas a 100 Mb/s, como ya se explicó en la introducción de este capítulo. Para ello, se ejecuta un mismo *script* con nombre *limit.sh* en ambas máquinas. En él se emplea la herramienta *tc*, incluida en *iproute2*, que como su nombre indica (*traffic control*) se encarga de controlar el tráfico en la red mediante la adición de reglas y disciplinas de programación en las diferentes interfaces. Los comandos empleados se pueden ver en la figura 6.5, donde se observa cómo se elimina primero cualquier configuración que hubiese antes, se establece la base para hacer la configuración nueva, y por último se añade una clase que limite el *rate* de la interfaz correspondiente.

```
#!/bin/bash

# Elimina cualquier configuracion posterior
sudo tc qdisc del dev eth1 root
# Base para realizar la configuracion
sudo tc qdisc add dev eth1  root handle 1: htb default 1
# Adicion de limitacion de BW
sudo tc class add dev eth1  parent 1: classid 0:1 htb rate 100mbit

sudo tc qdisc del dev eth2 root
sudo tc qdisc add dev eth2 root handle 1: htb default 1
sudo tc class add dev eth2 parent 1: classid 0:1 htb rate 100mbit

sudo tc qdisc del dev eth3 root
sudo tc qdisc add dev eth3 root handle 1: htb default 1
sudo tc class add dev eth3 parent 1: classid 0:1 htb rate 100mbit
```

Figura 6.5: Limitación de Velocidad en las Interfaces de las Máquinas.

A continuación, se puede proceder a realizar las pruebas. Para ello, se utilizan dos *scripts* bajo el mismo nombre *test_iperf3.sh* en cliente y servidor. Como se observa en las figuras 6.6 y 6.7, el servidor se pone en escucha, mientras que en el cliente se hace una conexión a la dirección 10.1.1.1 del servidor (opción -c *server*) durante 600 segundos (opción -t *duration*), guardando los resultados del *throughput* en un fichero de texto (opción *–logfile RESULT*) con formato *json* (opción -J).

```
#!/bin/bash

iperf3 -s
```

Figura 6.6: Test con *Iperf3* en el Servidor (*mptcpUe1*).

```
#!/bin/bash

server="10.1.1.1"
duration=600
if [[ -z "$1" ]]; then
    echo ""; echo "Using result.txt ..."; echo "";
    RESULT="result_... .txt"
else
    RESULT=$1
fi
iperf3 -c ${server} - J --logfile ${RESULT} -t ${duration}
```

Figura 6.7: Test con *Iperf3* en el Cliente (*mptcpUe2*).

Ejecutando dichos archivos en ambas máquinas, junto con *start_stats.sh* en *mptcpUe1* para enviar las estadísticas a *Grafana*, se hacen las pruebas:

DEFAULT SCHEDULER

Si se hace una prueba con el *default scheduler*, se obtiene la figura 6.8:

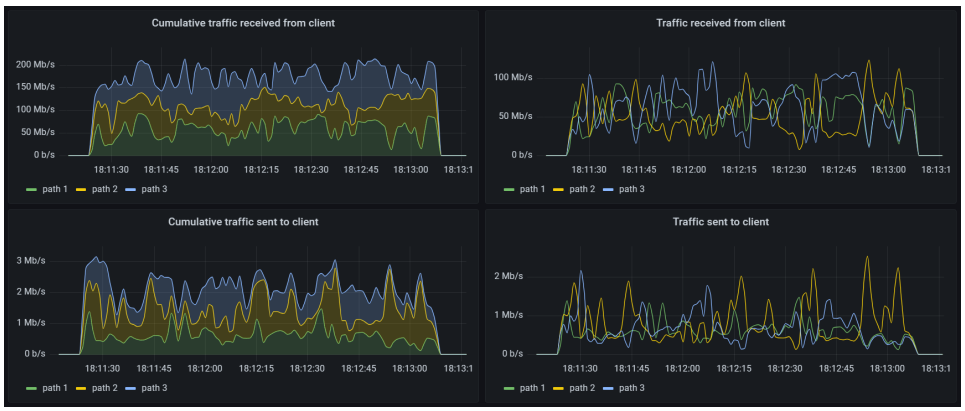


Figura 6.8: Comportamiento del *Default Scheduler* en MPTCP Versión 0.

Como se observa en la figura 6.8, cada vez se transmite en uno de los caminos o interfaces, esto es, en el subflujo que tenga menor RTT. Además, el tráfico recibido del cliente, que es el que manda los datos al servidor, llega a los 100 Mb/s, que es la cota máxima que se ha establecido por interfaz, y por ende, por subflujo (esto hace que se sepa que sólo se está transmitiendo en un subflujo, ya que de lo contrario se vería un tráfico de unos 100Mb/s x 3 subflujos=300 Mb/s). Por otro lado, el servidor sólo manda ACK, por lo que llega a una tasa de bits exitosos menor, de unos 3 Mb/s en acumulado. Así, se puede concluir cómo el *scheduler* por defecto cumple con lo previsto, no de forma ideal, ya que en dicho caso cuando se transmitiese por uno de los subflujos el resto debería estar a 0, lo cual no sucede tan exactamente, pero sí cumple con su función en carácter general.

Si se añadiese retardo en las interfaces con *tc*, se debería ver de forma más clara dicho funcionamiento. Por ejemplo, si se añade un retardo de 50 ms en *eth2* (*path* 2) y de 100 ms en *eth3* (*path* 3), y posteriormente unos 200 ms en *eth1* (*path* 1) con los siguientes comandos:

```
sudo tc qdisc add dev eth2 parent 1:1 netem delay 50ms
sudo tc qdisc add dev eth3 parent 1:1 netem delay 100ms
sudo tc qdisc add dev eth1 parent 1:1 netem delay 200ms
```

En la figura 6.9 se puede observar como el comportamiento es el esperado. En la recepción de los datos por parte del cliente, se transmite por el *path* 1, es decir, por uno de los subflujos de la interfaz *eth1* en unos 100 Mb/s (acontecimiento 1), hasta que se le añaden los 200 ms a esta interfaz; en dicho momento se deja de utilizar, y se empieza a emplear *eth2*, el *path* 2, ya

que ahora es la que tiene menos retardo de las tres (acontecimiento 2). En cuanto a los datos enviados al cliente ocurre lo mismo, pero el *throughput* es menor, de unos 5 Mb/s acumulados (sólo envío de ACK).

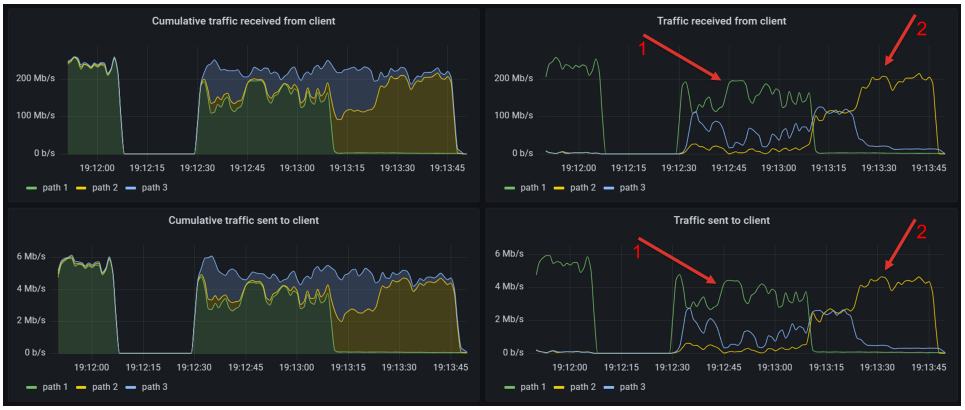


Figura 6.9: Comportamiento del *Default Scheduler* con Retardos Añadidos en MPTCP Versión 0.

Si no se limita la velocidad de las interfaces, y se añade un retardo de 100 ms a *eth2* y *eth3*, el comportamiento es más limpio, debido a que a veces cuando se hace limitación de velocidad en las interfaces, el funcionamiento de MPTCP cambia, ya que no está 100x100 pulido de errores. En la figura 6.10 se puede ver esto, se emplea únicamente el subflujo con menos RTT, el cual utiliza la interfaz *eth1* a 150 Mb/s (al no haber limitación de velocidad en este caso) ya que es la que no tiene retardo añadido.

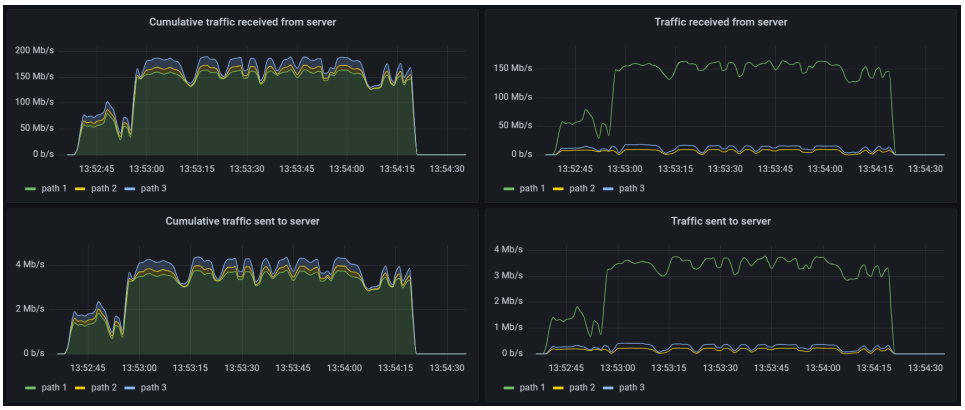


Figura 6.10: Comportamiento del *Default Scheduler* sin Limitación de *Throughput* y con Retardos Añadidos en MPTCP Versión 0.

REDUNDANT SCHEDULER

La figura 6.11 muestra su comportamiento:

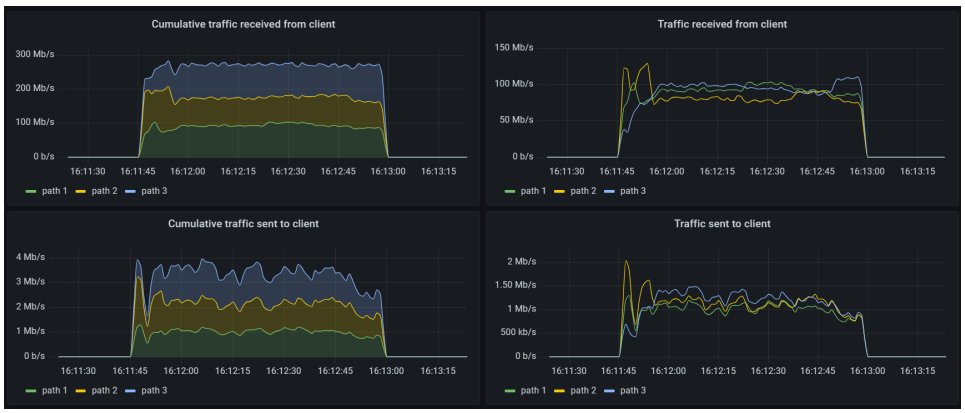


Figura 6.11: Comportamiento del *Redundant Scheduler* en MPTCP Versión 0.

Volviendo a la configuración inicial, con limitación de velocidad en las interfaces, se observa en comportamiento del *redundant scheduler* en la figura 6.11. En cuanto al tráfico recibido del cliente, se tiene que por las 3 interfaces hay un *throughput* de unos 100 Mb/s. Esto es así ya que por los diferentes subflujos que se establecen en cada interfaz, llegan los mismos datos a 100 Mb/s en cada uno, entonces la interfaz descarta todos (al estar repetidos) excepto los de uno de ellos, observándose así como la tasa máxima de cada interfaz es la de un único subflujo, unos 100 Mb/s (las pequeñas diferencias entre subflujos se deben a ciertas pérdidas que pueda haber). Si se estuviesen enviando diferentes datos, llegarían 100 Mb/s de cada subflujo, por lo que al utilizar un *path manager fullmesh* y haber subflujos entre cada par de direcciones IP, se verían 300 Mb/s en cada interfaz, al no descartarse nada. Si se añade un retardo de 100 ms en *eth2* y *eth3*, se obtendría la figura 6.12:

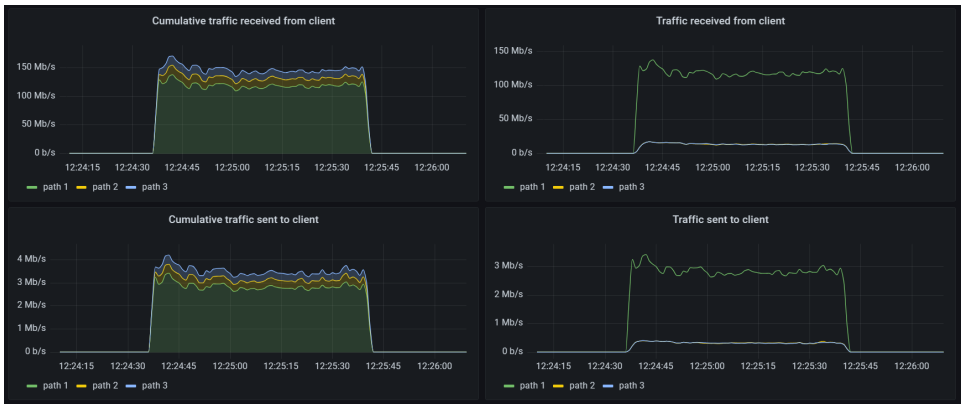


Figura 6.12: Comportamiento del *Redundant Scheduler* con Retardos Añadidos en MPTCP Versión 0.

En este caso, únicamente se transmiten los datos por uno de los subflujos asociados a la interfaz con menos RTT, la *eth1*. La razón de este comportamiento puede deberse a que una vez los datos han llegado al servidor por dicho subflujo, su confirmación llega al cliente antes que su envío por el resto de subflujos (al tener estos más retardo), por lo que al estar los datos repetidos, la aplicación decide no enviarlos de nuevo.
Por otro lado, el *throughput* alcanzado es de unos 120 Mb/s, ya que es posible que tanto haya ciertos sobrepasos de la cota establecida, como que no se llegue al alcanzar debido a limitaciones del portátil.

ROUND-ROBIN SCHEDULER

Como se vio en la sección 5.2.3, el *scheduler round-robin* manda los datos por los diferentes subflujos de manera *round-robin*, es decir, envía el primer paquete por el primer subflujo, el segundo por el siguiente subflujo, y así de manera circular. Pues bien, hay una opción que se puede configurar en este *scheduler* y que cambia la forma en el que éste se comporta. Si se establece con CWND_LIMITED=Y, se mandará un paquete por un subflujo, luego por el siguiente, y así; pero si uno de ellos tiene su ventana llena, se pasará al siguiente subflujo. Por ello, el *scheduler* no se comportará como un *round-robin* real. En caso de que se configure con CWND_LIMITED=N, se mandará un paquete en cada subflujo de igual manera, pero si uno de ellos tiene la ventana de congestión llena, no se pasa al siguiente subflujo, sino que se espera a que se vacíe algo la ventana para poder mandar el paquete por él. Por eso es round-robin perfecto, siempre manda un paquete por subflujo.

En las figuras 6.13 y 6.14 se muestran los resultados de este *scheduler* en ambos casos:

Figura 6.13: Comportamiento del *Round-robin Scheduler* con CWND_LIMITED=Y en MPTCP Versión 0.

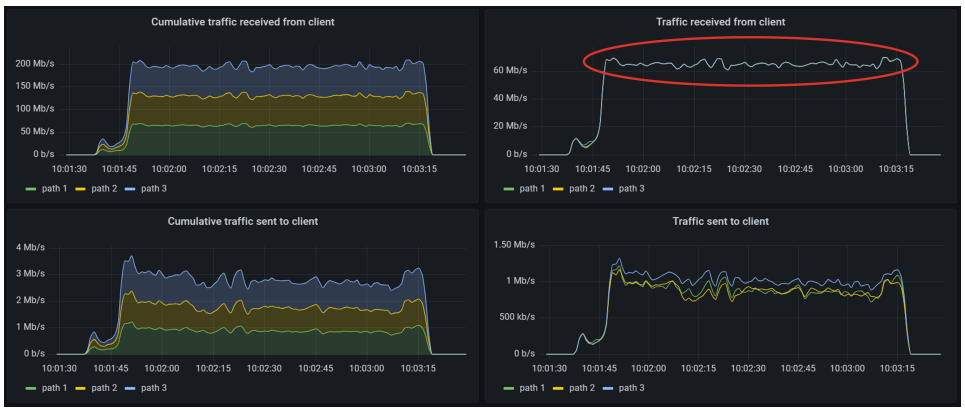


Figura 6.14: Comportamiento del *Round-robin Scheduler* con CWND_LIMITED=N en MPTCP Versión 0.

El resultado es bastante similar con un *throughput* un poco menor de los 100 Mb/s debido a las limitaciones del portátil. Sin embargo, en el caso de CWND_LIMITED=Y hay una variación un poco más grande entre subflujos por cómo se reparten los paquetes cuando algún subflujo está limitado instantáneamente por su ventana de congestión. Las diferencias más significativas se rodean en rojo en la figura 6.13. En el caso de CWND_LIMITED =N, al mandarse un paquete por subflujo, el *throughput* de cada interfaz corresponde a unos 100 Mb/s (se dan unos 70 Mb/s por motivos de potencia del portátil) y es prácticamente idéntico (en la figura 6.14 no se distingue un camino de otro en lo rodeado en rojo), ya que el comportamiento *round-robin* se cumple.

Como se ha podido ir viendo, los diferentes *schedulers* cumplen de una manera más o menos exacta con el comportamiento que se esperaba. Para finalizar esta verificación, se han procesado los datos que se guardaron con *–logfile RESULT* de las diferentes pruebas, utilizando el *script* de la figura 6.15.

A este *script* se le introduce como parámetro de entrada el fichero que se quiere procesar, y devuelve un fichero nuevo procesado con el mismo nombre que tenía el original pero con la palabra *processed* al final. Como se estableció que los resultados de *throughput* de las pruebas se almacenasen en formato *json*, se conoce donde se encuentran los datos que se quieren extraer, esto es, el valor de la variable *bits_per_second* de cada *stream*. De esta forma, se obtiene un archivo que contiene únicamente una lista de valores de *throughput*.

```
#!/bin/bash

FULLFILE=$1
filename=$(basename -- "$FULLFILE")
extension="${filename##*.}"
filename="${filename%.*}"

if [[ -z "$1" ]]; then
    echo ""; echo "Syntax: $0 <filename>"; echo ""
    exit 0
else
    processedfile="${filename}_processed.txt"
    echo ""; echo "Processing file ${filename}... Resulting file is ${
        processedfile}"; echo ""
    cat ${FULLFILE} | grep sum -A 6 | grep bits_per_second | cut -d":"
        -f 2 | cut -d "," -f 1 | tr -d "[:blank:]" > "${processedfile}"
fi
```

Figura 6.15: *Script* para el Procesamiento de los Datos Obtenidos en las Pruebas.

Una vez se procesan todos los resultados, se emplea un *script* de *MatLab* que se encarga de procesar los datos y elaborar una gráfica con las CDF de cada *scheduler*. Este *script* se puede ver en el Apéndice C, y da lugar a la figura 6.16:

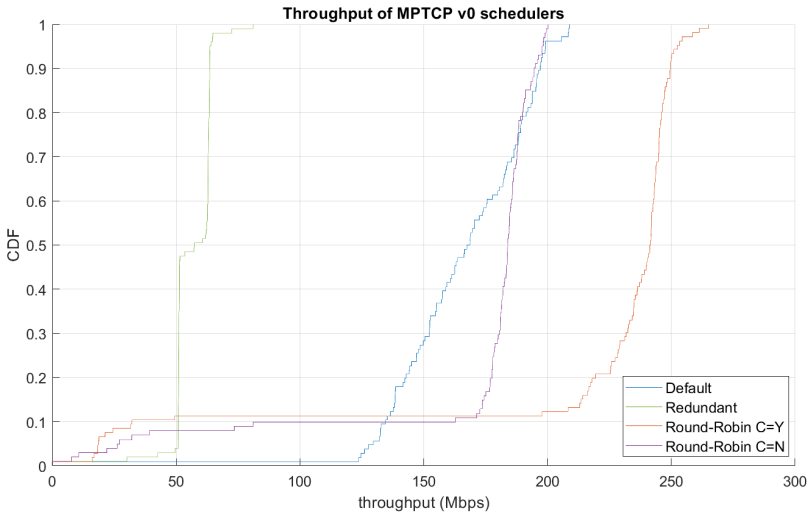


Figura 6.16: CDF de cada *Scheduler* en MPTCP Versión 0.

Según lo que se puede extraer de la figura 6.16, se puede afirmar como el *redundant scheduler* es quizás el que tiene peor comportamiento, debido a que el *throughput* no llega a estabilizarse ni en 100 Mb/s. El *default scheduler* tiene una CDF más exponencial con una probabilidad mayor de que el *th-*

roughput sea más elevado de 100 Mb/s, incluso llega a los 200 Mb/s. Por su lado, el *round-robin scheduler* en sus dos versiones tiene un comportamiento similar. Con CWND_LIMITED=Y se alcanza un *throughput* mayor debido a que no hay esperas en la transmisión; en el caso de CWND_LIMITED=N, el *throughput* es menor debido a que cumple a la perfección la transmisión consecutiva de un paquete por subflujo, teniendo que 'pararse' y no saltar al siguiente cuando alguno tenga la ventana de congestión llena. Aun así, los valores son más estables que en el caso contrario.

6.3.1. Escenario Real con 5G y Wi-Fi

Para poder acercar este proyecto un poco más a la realidad, se ha creado y configurado un escenario real en el que se usa MPTCP versión 0 para la comunicación por 5G y Wi-Fi. Para ello, se han utilizado una serie de equipos disponibles en la Universidad y se ha elaborado el escenario de la figura 6.17:

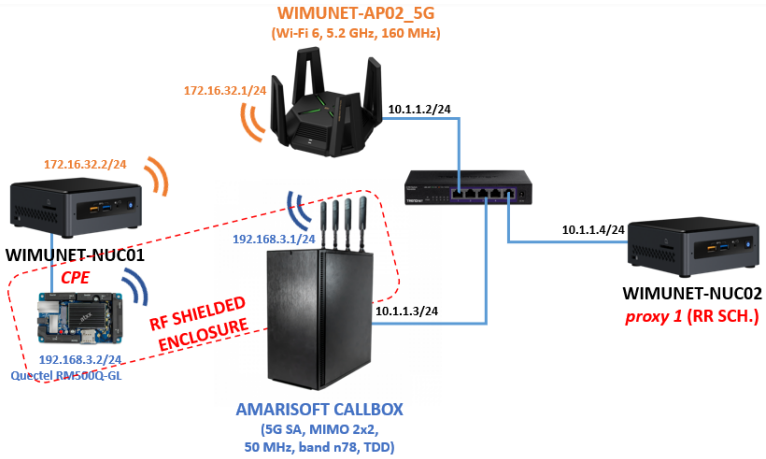


Figura 6.17: Esquema del Escenario MPTCP Versión 0 con 5G y Wi-Fi [96].

El escenario cuenta con los siguientes elementos:

- **2 Intel NUC** con i7 + 16 GB de RAM + 512 GB de SSD. Uno hace de cliente (WIMUNET-NUC02) y otro de servidor (WIMUNET-NUC3).
- **Un punto de acceso Wi-Fi 6**: Xiaomi AX9000 con dirección IP actual para la gestión: 192.168.32.1/24.
- **Una estación base Amarisoft** con 4G, 5G *Non-Stand Alone* (NSA) y 5G *Stand Alone* (SA). Para este escenario se usa 5G SA, banda n78 con *Time Division Duplexing* (TDD) y un ancho de banda 50 MHz.
- **Un módem 5G** (Quectel RM500Q) para la conexión a la red 5G. Este se incorpora en el NUC cliente y juntos forma el *Customer Provided Equipment* (CPE).

- **Un *switch*** para encaminar tráfico.
- **Un recinto blindado de RF**, es decir, una jaula de Faraday, para que las antenas del *gNodeB* (gNB) y el módem 5G conectado al CPE estén en el interior, evitando así radiar en bandas de frecuencia que requieran licencia.

Además de todos ellos, se emplea un PC (monitor) para poder visualizar los resultados en *Grafana*. Este escenario montado en la Universidad se puede ver en la figura 6.18:

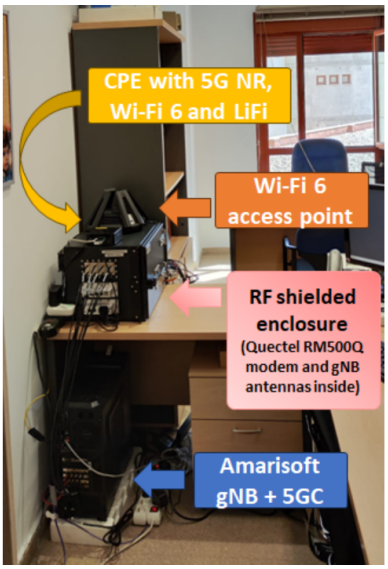


Figura 6.18: Montaje del Escenario Real con 5G y Wi-Fi [96].

Lo siguiente que se debe hacer es configurar el sistema, partiendo de que los dispositivos ya cuentan con las direcciones IP que se muestran en la figura 6.17 asignadas. Los pasos que se han seguido se describen a continuación:

1. Lo primero que se debe hacer es iniciar la estación base y configurarla. Para ello se utiliza un archivo de configuración *.cfg* que se encarga de iniciarla y configurar los parámetros necesarios. Seguidamente, mediante el comando *service lte start* se puede iniciar. En el caso del punto de acceso Wi-Fi, siempre está activo para recibir conexiones.

2. Una vez se tiene la estación base 5G y el punto de acceso Wi-Fi funcionando, se deben configurar cliente y servidor. Para ello, se utilizan los mismos comandos que se vieron en la sección 5.2.3 (en este caso se va a hacer una prueba con el *path manager fullmesh*, control de congestión olia, y *round-robin scheduler*). Esto se puede ver en las figuras 6.19 y 6.20:

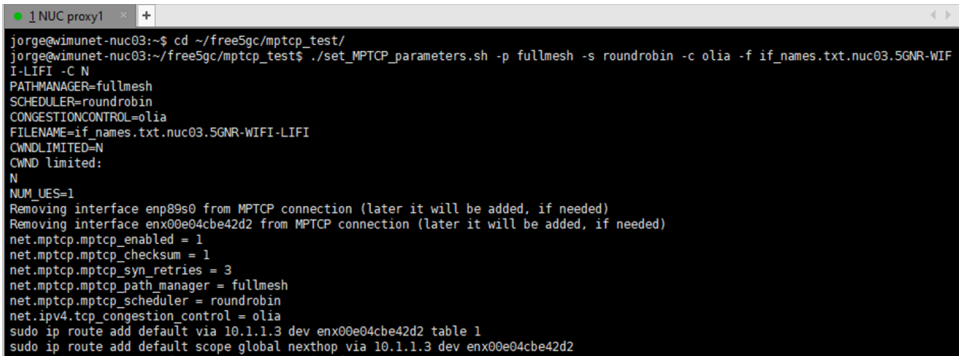


Figura 6.19: Configuración del *Routing* y de MPTCP Versión 0 en el Servidor.

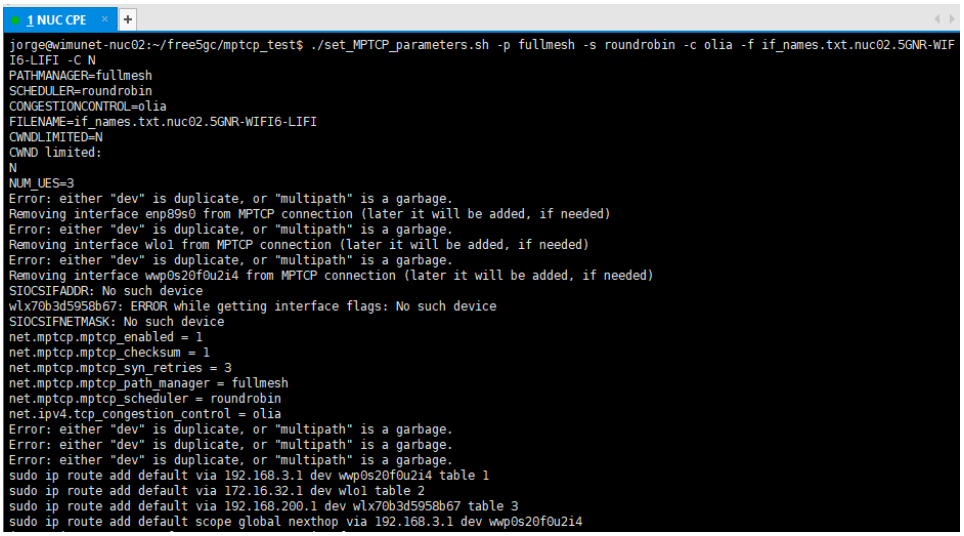


Figura 6.20: Configuración del *Routing* y de MPTCP Versión 0 en el Cliente.

Cabe destacar que ahora el fichero con el direccionamiento proporcionado mediante la opción -f ya no es el mismo que se usaba en MPTCP versión 0 virtualizado, ya que ahora las direcciones han cambiado.

3. Lo siguiente que se debe establecer es la conexión entre el CPE y la estación base 5G. Para ello, se utiliza el *script connect_quectel_amarisoft.sh* en el CPE, siempre y cuando el servicio proporcionado por la estación base esté activo y corriendo. Tanto la activación del servicio 5G, como la conexión del CPE a la estación base, se pueden ver en las figuras 6.21 y 6.22:

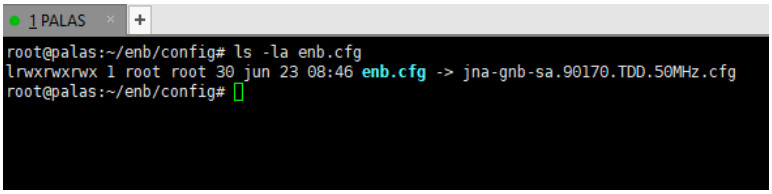


Figura 6.21: Puesta en Marcha de la Estación 5G.

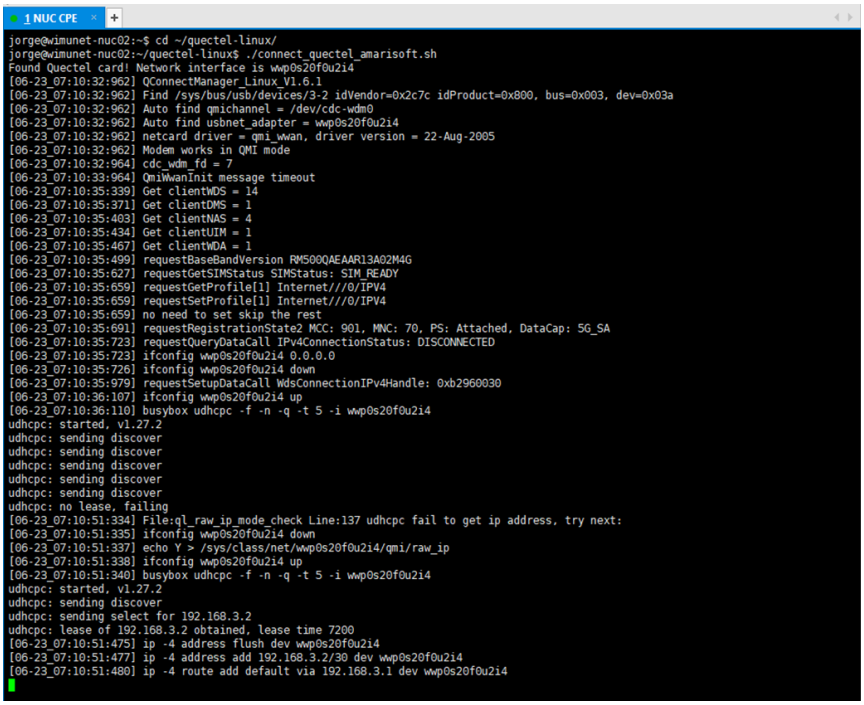


Figura 6.22: Conexión del CPE a la Estación 5G.

Un concepto a mencionar independiente a MPTCP, tiene que ver con la estación base 5G.

Amarisoft es una empresa de software especializada en equipos 4G y 5G, la cual proporciona soluciones eficaces y de gran alcance para los problemas de red que se dan en los sistemas de comunicaciones actuales. La estación base utilizada en este escenario es de esta empresa, y proporciona una interfaz de usuario o *Graphical User Interface* (GUI), la cual permite monitorizar el rendimiento de la red, ver los mensajes que se han enviado y recibido en la estación, realizar pruebas, etc. En la figura 6.23 se puede ver esta interfaz, con los diferentes mensajes que se han ido mandando durante la inicialización y configuración de la estación.

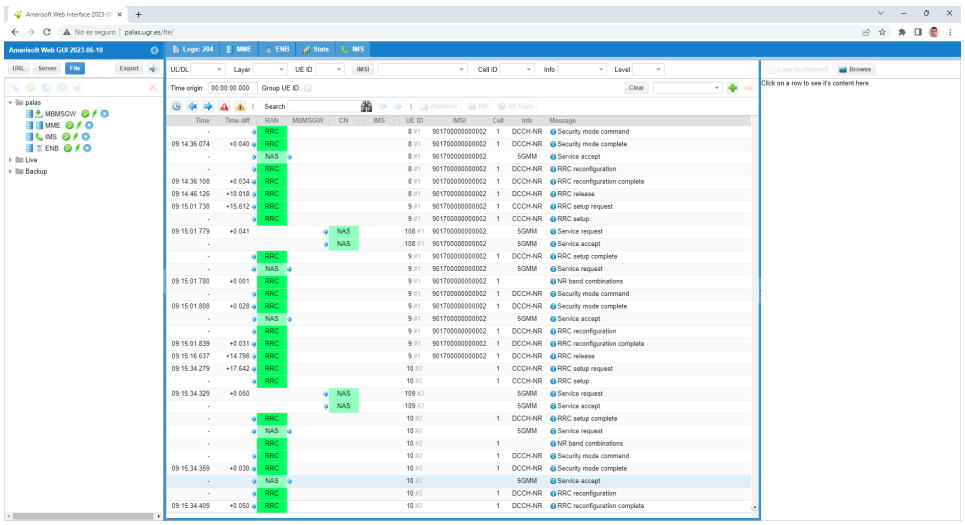


Figura 6.23: GUI de Amarisoft.

Con todo configurado, se hace una conexión *iperf3* entre cliente y servidor. Previo a esto, en el cliente se ha debido configurar el envío de estadísticas a *Grafana* siguiendo los pasos vistos en la sección 6.2. Los comandos empleados son los siguientes:

```
iperf3 -s (en el servidor)
iperf3 -c 10.1.1.4 -t 60 -R (en el cliente)
```

La opción -R en el cliente indica que va a ser el servidor el que mande paquetes al cliente, no al revés. Esto se hace así debido a que la red 5G tiene un *throughput* mucho mayor en el sentido descendente que en el ascendente. En Wi-Fi no hay problema, es simétrico.

Accediendo a *Grafana* en el monitor conectado al CPE, se observa el comportamiento mostrado en la figura 6.24:

Figura 6.24: Comportamiento de MPTCP Versión 0 con el *Round-robin Scheduler* y usando 5G y Wi-Fi.

En sentido cliente-servidor, en la figura 6.24 se puede ver que tanto Wi-Fi como 5G dan una tasa de tráfico de unos 250 Mb/s aproximadamente. La conexión 5G debería dar unos 350 Mb/s, y la conexión Wi-Fi 1 Gb/s aproximadamente, por lo que los subflujos que usan esta última tecnología experimentan menor RTT. Esto no sucede así debido a un fenómeno denominado *head-of-line blocking*.

El protocolo MPTCP garantiza la entrega ordenada de los paquetes en el receptor a la aplicación, y como se acaba de mencionar, es posible que haya paquetes que lleguen antes (por Wi-Fi) que otros (por 5G). Si se da el caso en el que un paquete no ha llegado y los paquetes que deberían haber llegado tras él, ya lo han hecho, estos tendrán que esperar, asegurando así que la entrega se hace de forma ordenada. De esta manera, un subflujo lento afecta al rendimiento de los subflujos rápidos. Este bloqueo tiene un gran impacto negativo en el rendimiento del *round-robin* cuando las tecnologías inalámbricas tienen características diferentes.

En sentido contrario, el comportamiento es el mismo, pero el cliente solo confirma la recepción de los datos, por lo que el *throughput* es menor.

6.4. *Throughput* en MPTCP Versión 1

Para realizar las pruebas en el escenario virtual con MPTCP versión 1, se configura el *path management* mediante *ip mptcp* poniendo todas las interfaces del cliente como *subflow fullmesh* para que creen subflujos con todas las interfaces del servidor, y las de éste como *signal* para que se anuncien sus direcciones; de esta forma se crea la misma topología *fullmesh* que se obtuvo en MPTCP versión 0, simplemente variando el tipo de *scheduler* empleado. A continuación, se vuelven a limitar las interfaces de ambas máquinas a 100 Mb/s con el *script* mostrado en la figura 6.5 (cambiando el nombre de las interfaces por *enp0s8*, *enp0s9* y *enp0s10*). Finalmente, se crea un nuevo *dashboard* en *Grafana* que muestra 4 gráficas relacionadas con el tráfico normal y el tráfico acumulado enviado y recibido del cliente en las 3 interfaces disponibles.

Una vez se tiene lo anterior configurado, para hacer las pruebas se emplean también los *scripts* de las figuras 6.6 y 6.7, pero ahora no es la máquina 1 la que hace de servidor, sino la segunda máquina, por lo que la conexión que hace el cliente ya no es a la dirección 10.1.1.1 sino a la 10.1.1.4. De esta forma, con MPTCP configurado y las limitaciones impuestas en las interfaces, se ejecutan los *scripts* de prueba en ambas máquinas, junto con *start_stats.sh* en la máquina *server*, y se comienzan a realizar las pruebas con los diferentes *schedulers*:

DEFAULT SCHEDULER

En esta versión del protocolo, el *scheduler* por defecto no transmite por el subflujo con menos RTT, sino que funciona de la misma manera que el *redundant*, transmite los mismos datos por todas las rutas disponibles, con el fin de buscar el máximo rendimiento. Al hacer una prueba con este *scheduler*, se han obtenido los resultados mostrados en la figura 6.25:

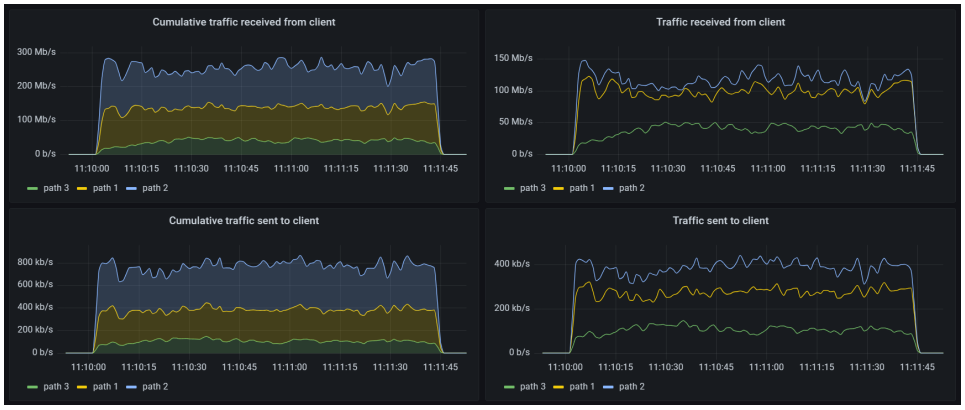


Figura 6.25: Comportamiento del *Default Scheduler* en MPTCP Versión 1.

Como se observa en la figura 6.25, en cuanto al tráfico recibido del cliente, las interfaces *enp0s8* y *enp0s9* (*path* 1 y *path* 2) dan unos 100 Mb/s, mientras que *enp0s10* da unos 50 Mb/s (el protocolo no utiliza al 100 % dicha interfaz); por lo demás, al dar una tasa de 100 Mb/s se comprueba que los mismos datos están siendo transmitidos por todos los subflujos, descartando aquellos repetidos. En acumulado se obtiene un *throughput* de 300 Mb/s aproximadamente.
El tráfico en la dirección contraria tiene un comportamiento similar y está formado únicamente por mensajes ACK, por lo que la tasa es menor, de unos 800 kb/s en acumulado.

FIRST SCHEDULER

Este *scheduler* se incorpora nuevo en la versión 1 de MPTCP. Su función es mandar los datos por el primer subflujo disponible, que como cabe esperar, siempre es por el que se hace la conexión. Esto se puede corroborar en la figura 6.26, donde se obtienen unos 80 Mb/s por el *path* 1, esto es, por el primero de los subflujos asociados a la interfaz *enp0s8*, por lo tanto, el subflujo de conexión entre 10.1.1.3 y 10.1.1.4. Al mismo tiempo, se tiene una tasa de 200 kb/s de servidor a cliente usando el mismo subflujo.

Figura 6.26: Comportamiento del *First Scheduler* en MPTCP Versión 1.

BACKUP SCHEDULER

Otro *scheduler* que se incorpora nuevo es el *backup scheduler*. Éste manda los datos por el primer subflujo que no está en *backup*. Al analizar esto, se puede deducir que su comportamiento va a ser el mismo que el del *first*, ya que el primer subflujo que nunca va a estar en *backup* es por el que se hace la conexión, el primero, por lo que independientemente del estado que tengan los demás subflujos, siempre se enviarán los datos por él. Esto se puede comprobar observando la figura 6.27:

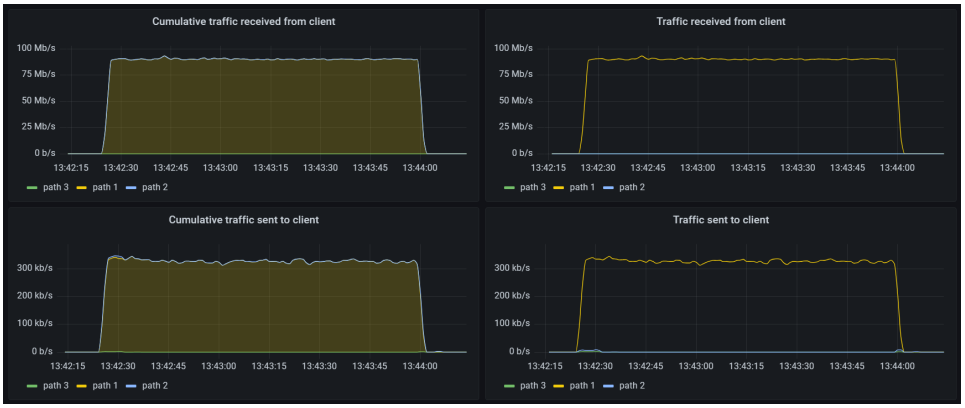


Figura 6.27: Comportamiento del *Backup Scheduler* en MPTCP Versión 1.

ROUND-ROBIN SCHEDULER

Con este *scheduler* no se han podido llevar a cabo pruebas debido a que actualmente, a fecha 8 de junio de 2023, no está implementado correctamente y provoca bloqueos en los dispositivos que hacen la comunicación. Se prevé

su estudio en estos meses posteriores, pero al tener que modificar funciones y código de numerosos archivos del *kernel*, está llevando tiempo. En caso de que estuviera disponible, en las gráficas se observaría cómo siempre elige el siguiente subflujo disponible para enviar datos. Si no hay subflujo disponible, elige el primero.

REDUNDANT SCHEDULER

El último *scheduler* que se ofrece en esta versión es el *redundant*. Éste funciona igual que el *default scheduler* en esta versión, y que el *redundant scheduler* en MPTCP versión 0. Su objetivo es enviar todos los datos por todas las rutas disponibles.

Al hacer una prueba con él, se puede ver como el tráfico recibido del cliente por todas las interfaces es de unos 100 Mb/s, unos 300 Mb/s en acumulado. Esto verifica su correcto funcionamiento.
A la hora de mandar datos al cliente por parte del servidor, se obtiene un *throughput* acumulado de unos 1.5 Mb/s. Este valor es bajo ya que como se ha explicado en las pruebas anteriores, este tráfico corresponde únicamente a mensajes ACK de confirmación. Todo esto se ve reflejado en la figura 6.28:

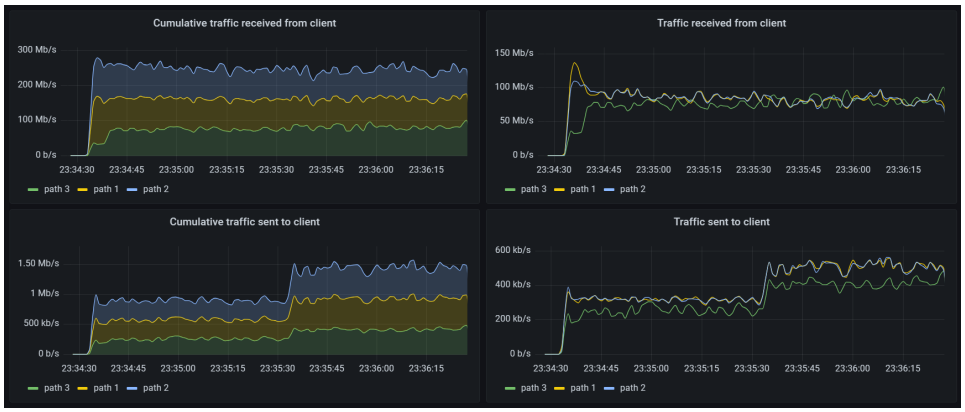


Figura 6.28: Comportamiento del *Redundant Scheduler* en MPTCP Versión 1.

Si con la herramienta *tc* se añade un retardo de 100 ms a las interfaces *enp0s9* y *enp0s10* (*path* 1 y *path* 2), el comportamiento que tiene el *scheduler* es el reflejado en la figura 6.29. Los datos utilizan un subflujo de la interfaz *enp0s8* (*path* 1) con un *throughput* de 100 Mb/s aproximadamente, mientras que en las otras interfaces la tasa es prácticamente nula.

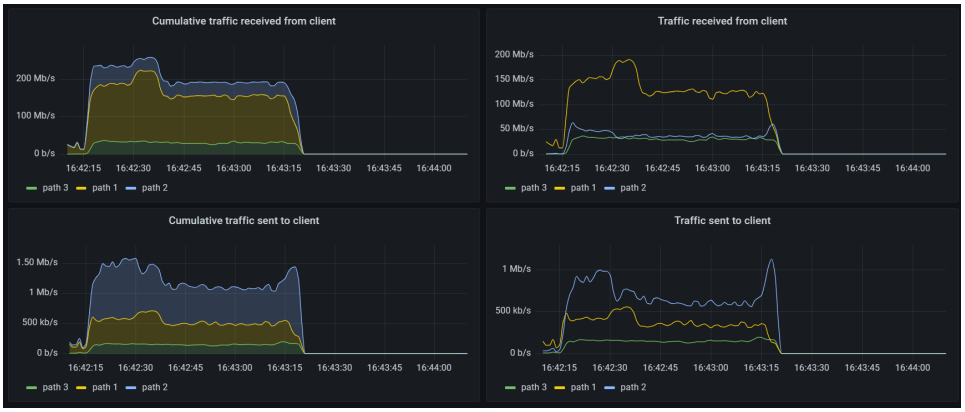


Figura 6.29: Comportamiento del *Redundant Scheduler* con Retardos Añadidos en MPTCP Versión 1.

Una vez se ha analizado el rendimiento de cada uno de los *schedulers*, al igual que se hizo con MPTCP versión 0, se emplea el *script* de la figura 6.15 para procesar los resultados obtenidos, y se obtiene la gráfica de la figura 6.30. Esta muestra las CDF de cada *scheduler* usando el *script* de *MatLab* mostrado en el Apéndice C (cabe destacar que los archivos empleados provienen de las pruebas simples con los *scheduler*, es decir, las pruebas únicamente con limitaciones de velocidad, no con retardos añadidos).

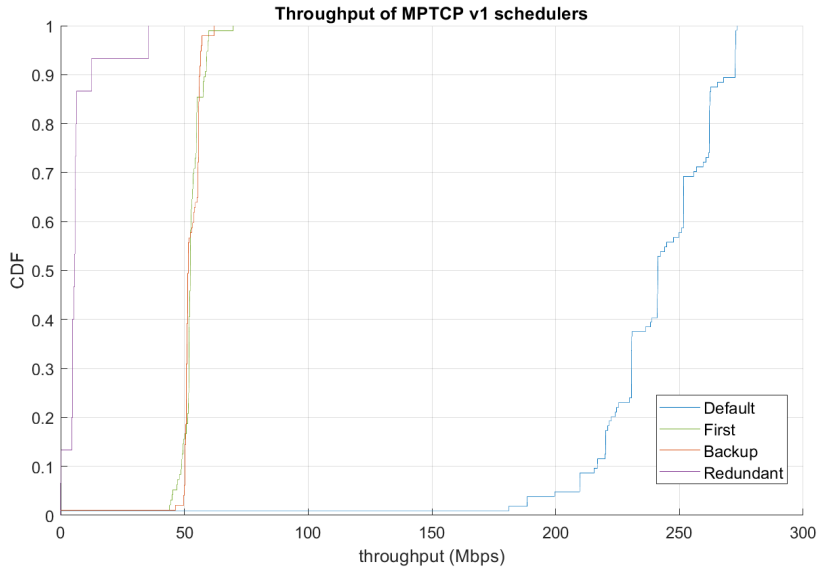


Figura 6.30: CDF de cada *Scheduler* en MPTCP Versión 1.

Como se observa en la figura 6.30, el *throughput* de los *schedulers first y backup* es prácticamente idéntico, entorno a los 50 Mb/s. Por otro lado, el *redundant* es el que peor rendimiento ofrece, al igual que ocurría en la versión 0. Finalmente, se observa el *default scheduler*, cuyo comportamiento es el más beneficioso, es el que ofrece el mayor *throughput* y además este crece de forma exponencial con una pendiente pronunciada, por lo que no suele haber momentos con valores bajos de *throughput*, lo que beneficia a la aplicación.

Con todo esto, queda descrito y verificado el comportamiento de MPTCP versión 0 y versión 1 en los escenarios virtuales creados. Además, se ha hecho una pequeña prueba de MPTCP versión 0 con un escenario real de 5G y Wi-Fi, por lo que se han cubierto las competencias esperadas, y además se han ampliado.

6.5. Complicaciones Encontradas en el Proyecto

Ahora que se ha explicado cómo se ha hecho la implementación de las dos versiones del protocolo en los escenarios virtuales creados, eligiendo la solución que más aportaba al proyecto, y configurando el escenario de experimentación para luego realizar las pruebas, se pueden comentar los impedimentos que se han ido encontrando en su desarrollo.

A la fecha de realización del proyecto, a lo largo del año 2022-2023, MPTCP versión 0 ya había sido estudiado e implementado por numerosas organizaciones, por lo que al estar bastante analizado, no contenía muchos errores o comportamientos anormales. Esto se ha podido ver reflejado en los resultados obtenidos de las pruebas.

A pesar de esto, el desafío enfrentado ha tenido sus raíces en MPTCP versión 1. Como ya se ha comentado, para llevar a cabo su implementación se utilizó el material provisto en el proyecto *Upstream* alojado en GitHub. Este proyecto empezó con el surgimiento de esta nueva versión del protocolo. Sin embargo, no ha tenido un plazo cerrado de trabajo y estudio, sino que ha estado y está en constante desarrollo. Es por esto que cuando se inició la parte del proyecto relacionada con su implementación, el código fuente del *kernel* estudiado tenía que ver con la versión 6.1, esta versión incluía: permitir operaciones privilegiadas de *Netlink* desde espacios de nombres de usuario, soporte TCP_FASTOPEN_CONNECT para que un cliente iniciase conexiones MPTCP con datos en los mensajes SYN, etc. Lo más relevante es que no se habían implementado los nuevos *schedulers* que se han comentado (*first, backup, redundant* o *round-robin*), únicamente estaba disponible el *scheduler* por defecto.

Con dicho *kernel* implementado, se estuvo en constante alerta al proyecto *Upstream*, ya que continuamente se iban alertando avances. Anterior a la realización de pruebas, surgió una nueva modificación de éste, la incorporación de diferentes tipos de *schedulers* mediante módulos eBPF cargados en el *kernel*, y probados con *auto-tests*. Esto hizo que se tuviese que remodelar toda la implementación, y comenzó el estudio de cómo poder cargar dichos *schedulers* y seleccionarlos para su uso. Posteriormente, alrededor de un mes después, una vez se había logrado conseguir cargar dichos *schedulers* en el *kernel* 6.1, surgió una nueva versión modificada, la versión 6.4-rc5+. Ésta no solo traía avances sobre el funcionamiento interno del protocolo, sino que además se incorporaba la elección de los diferentes tipos de *schedulers* mediante una variable *sysctl*.

Debido a que una de las intenciones de este proyecto es el estudio del protocolo mediante las soluciones más actuales y vigentes que hay hoy en día, se decidió volver a reestructurar la implementación y utilizar el nuevo *kernel*, con el fin de ofrecer un proyecto relevante con una base a futuro. Esta última actualización es la que se ha llevado a cabo y mostrado en esta memoria.

Todas estas modificaciones se han ido teniendo en consideración, y han hecho que el desarrollo del proyecto se haya visto entorpecido en alguna ocasión. A pesar de ello, estas complicaciones se han superado con éxito y se ha logrado elaborar un proyecto acorde a lo establecido, incluso con ciertas ampliaciones.

Capítulo 7

Conclusiones y Líneas Futuras

A continuación, se exponen las conclusiones obtenidas tras la realización del proyecto, junto con los posibles trabajos futuros a realizar.

7.1. Conclusiones

Este proyecto se ha ido desarrollando a la vez que una de las partes de las que trata, por lo que en su elaboración se han tenido que hacer frente a una serie de complicaciones motivadas por la actualización constante de las herramientas a emplear. Este proceso se describe en la sección 6.5.

En un principio se estudiaron diferentes proyectos internacionales que han hecho o hacen uso de MPTCP, con el objetivo de mostrar el gran alcance que tiene el protocolo. Entre ellos se mencionó un proyecto europeo del que forma parte la Universidad de Granada, y el cual se ha empleado en la elaboración de un escenario real como ampliación del proyecto. Seguidamente, se desarrollaron los conceptos más relevantes de MPTCP en sus dos versiones. Con esto se intentó dar una visión clara y esquemática de las diferentes funciones, mensajes y situaciones que se pueden dar en el Internet actual, y frente a las cuales este protocolo actúa con más ventajas que el TCP básico. Como se concluyó al final del capítulo 3, este protocolo siempre va a aportar ventajas ya que en el peor de los casos funcionará igual que TCP.

Previo a la implementación de los diferentes escenarios virtuales que se marcaron como objetivo en el capítulo 1, se llevó a cabo un resumen de la planificación y presupuestos del proyecto. Esto mostró una idea clara, aunque no del todo precisa, de los pasos que se fueron tomando. Una vez se marcó la línea temporal a seguir, han ido surgiendo ciertas variaciones de tiempo, debido a las actualizaciones continuas de MPTCP versión 1, y a la incorpo-

ración del caso real con 5G y Wi-Fi. Aun así, la planificación final que se ha seguido no desvaría mucho de la mostrada en el capítulo 4.

Pasando a la implementación de los escenarios virtuales, uno con MPTCP versión 0 y el otro con MPTCP versión 1, estos se explicaron en el capítulo 5. Para la versión 0, se utilizó la herramienta *Vagrant* y se tenía una guía con los pasos a seguir, por lo que resultó más sencillo. Para la versión 1 se tuvo que compilar un *kernel* modificado, añadir los diferentes *schedulers* como módulos en dicho *kernel*, investigar sobre qué funciones estaban disponibles en el momento de la implementación (ya que cada día surgía algo nuevo) y el por qué de los errores que ocurrían, etc. Elaborando ambos desarrollos, se pudo afirmar como tanto una implementación como otra, llevaban a la creación de un mismo escenario MPTCP, cada uno con unas funcionalidades dependientes de la versión del protocolo empleada.

Por último, se montó un escenario de pruebas bastante útil y visual, ya que gracias a *Grafana* se pudieron ver las diferentes gráficas con los resultados, simplemente con una serie de comandos y un archivo de configuración *json*. Además, con *MatLab* también resultó muy fácil elaborar las CDF de cada *scheduler*, para su posterior comparación.
Finalmente, se realizaron una serie de pruebas con resultados curiosos. En MPTCP versión 0 se obtuvo como el *default scheduler* o el *round-robin* con CWND_LIMITED=Y eran los que daban un *throughput* elevado. Por su lado, en MPTCP versión 1 no se obtuvo un comportamiento realmente adecuado de ninguno de ellos. Los *schedulers first* y *backup* eran similares, pero no muy ventajosos, mientras que el *redundant* tenía un comportamiento ciertamente extraño; el *default scheduler* era el que más se acercaba a lo deseado, maximizar el *throughput* agregado. Aun así, se pudo comprobar como el uso del protocolo trae consigo numerosas ventajas en cuanto a entrega fiable de paquetes (con el *redundant scheduler* en versión 0 y 1, y el *default scheduler* en v1, si los datos se pierden en un subflujo, pueden llegar por los demás), mejora de *throughput* y disminución de retardo (con el *default scheduler* en versión 0 se hace un uso eficiente de las rutas en función del RTT, para intentar alcanzar la mayor tasa de bits en el menor tiempo posible), etc.

Además de todas estas pruebas, se decidió aprovechar los equipos disponibles en la Universidad para la creación de un pequeño escenario con 5G y Wi-Fi. Esto se decidió así con el ánimo de acercar el proyecto un poco más a la realidad; además, en los escenarios virtuales se limitó la velocidad de todas las interfaces a 100 Mb/s, lo que no dejaba ver bien la capacidad que podía llegar a ofrecer el protocolo. Así, con este escenario se pudo observar como los rendimientos máximos en 5G y Wi-Fi se cumplían (con intervención en contra del *head-of-line blocking*), pudiendo comprobar como el uso de ambos

medios simultáneamente provoca un aumento del *throughput* obtenido en la red, así como una disminución del retardo y del periodo de indisponibilidad en caso de que haya pérdidas de conexión.

Con todo esto, se puede afirmar que los objetivos propuestos al inicio del proyecto se han cumplido satisfactoriamente, incluso han sido ampliados (con el estudio de la situación actual de MPTCP versión 1, y la creación del escenario real) con fines educativos futuros.

7.2. Líneas Futuras

A pesar de que se tienen los dos escenarios virtuales funcionando correctamente, hay ciertas mejoras que se pueden llevar a cabo todavía. En cuanto a MPTCP versión 0, no hay mucho que estudiar, quizás perfeccionar el funcionamiento de los *schedulers*, pero está bastante bien implementado. Por otro lado, MPTCP versión 1 sí que necesita unos años más de desarrollo. Entre los avances que se pueden llevar a cabo y se están planteando, se encuentran:

- **Mejoras en cuanto a *schedulers***: como se ha podido ver en las pruebas realizadas en el capítulo 6, los *schedulers* no funcionan a la perfección, incluso alguno de ellos, como el *round-robin*, no se puede utilizar. Dependiendo de la situación en la red, pueden tener ciertos comportamientos imprevistos y no deseados. Ahora mismo son una herramienta de prueba, pero en los próximos meses y años se prevé que se perfeccionen.

- **Más seguridad en el uso del protocolo**: otro concepto que sería bueno implementar y que se está estudiando es el uso del protocolo TLS junto con MPTCP, con el fin de aportar más seguridad.

- **Funcionalidades extendidas en el demonio *mptcpd***: se debería estudiar la adición de más variables en *mptcpd* para que permita poner los *endpoints* de un dispositivo con diferentes *flags*, ya que por el momento, solo se puede poner la misma *flag* para todos, es decir, todos como *subflow*, como *signal*, etc. De esta forma, se podría 'jugar' más con la configuración de MPTCP y probar más tipos de situaciones.

- **Aplicación para el uso de *sockets* MPTCP en lugar de *mptcpize***: este concepto no es algo que sea prioridad para los mantenedores del proyecto *Upstream*, ya que la herramienta *mptcpize* realiza las funciones requeridas; sin embargo, si sería conveniente estudiarla en caso de que en un futuro se requiera el uso de *sockets* MPTCP en todas las aplicaciones, y ya no sea de elección.

- **Adición de nuevas funciones al protocolo para cumplir el RFC 8684 [3]**: además de los objetivos que se acaban de mencionar, también existen muchas líneas abiertas en las cuales los mantenedores están trabajando. Estas incluyen la adición de más funcionalidades al protocolo, hasta el punto de cumplir a la perfección lo establecido en el RFC 8684 [3]. Entre algunas de estas mejoras se encuentran: la inclusión de la funcionalidad *backup* en los *schedulers* BPF, al igual que está actualmente en el *scheduler* por defecto, optimización de la API para que sea más fácil de manejar, mejora del etiquetado de subflujos, soporte para *kernels* antiguos, etc.

Independientemente de los planes futuros que se acaban de comentar, también resultaría más que interesante hacer una prueba con MPTCP versión 1 en equipos reales. El grupo de investigación de la Universidad ya trabajaba con la versión 0, por lo que montar el escenario para la prueba que se ha presentado en este proyecto resultaba relativamente rápido. Si se quisiesen hacer pruebas con dichos equipos implementando MPTCP versión 1, habría que volver a gestionar y configurar los equipos desde 0. Por motivos de tiempo no se realizó, pero es algo que se tiene pensado a futuro.

Bibliografía

[1] Miller, M. (2015). *The internet of things: How smart TVs, smart cars, smart homes, and smart cities are changing the world.* Indianapolis, Indiana. Pearson Education. [En línea]. Consultado el 30 de abril de 2023, desde http://uru.ac.in/uruonlinelibrary/Internet_of_Things/IOT%20How%20and%20Why.pdf

[2] Ford, A., Raiciu, C., Handley, M. J., & Bonaventure, O. (2013, enero). *TCP Extensions for Multipath Operation with Multiple Addresses* (Request for Comments N.º 6824). Internet Engineering Task Force. doi: 10.17487/RFC6824. Consultado el 30 de abril de 2023, desde https://datatracker.ietf.org/doc/rfc6824

[3] Ford, A., Raiciu, C., Handley, M. J., Bonaventure, O., & Paasch, C. (2020, marzo). *TCP Extensions for Multipath Operation with Multiple Addresses* (Request for Comments N.º 8684). Internet Engineering Task Force. doi: 10.17487/RFC8684. Consultado el 30 de abril de 2023, desde https://datatracker.ietf.org/doc/rfc8684

[4] Bonaventure, O. (2014, febrero). *The first ten years of Multipath TCP — MPTCP* [Imagen]. Consultado el 30 de abril de 2023, desde http://blog.multipath-tcp.org/blog/html/2018/12/01/multipath_tcp_a_retrospective.html

[5] PACKET593. (2017, junio). *¡¿Qué es Multipath TCP?!.* Packet593. [Imagen]. Consultado el 30 de abril de 2023, desde https://packet593.wordpress.com/2017/06/16/que-es-multipath-tcp/

[6] Bonaventure, O., Paasch, C., & Detal, G. (2017, enero). *Use Cases and Operational Experience with Multipath TCP* (Request for Comments N.º 8041). Internet Engineering Task Force. doi: 10.17487/RFC8041. Consultado el 7 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc8041

[7] Apple Support. (2022, febrero). *Usar Multipath TCP para crear conexiones de respaldo para iOS.* Consultado el 7 de mayo de 2023, desde https://support.apple.com/es-cl/HT201373

[8] Nguyen, K., Ji, Y., & Yamada, S. (2013). Improving WiFi networking with concurrent connections and multipath TCP. En *2013 IEEE 14th International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pp. 1-3. [En línea]. doi: 10.1109/WoWMoM.2013.6583365. Consultado el 7 de mayo de 2023, desde https://ieeexplore.ieee.org/document/6583365.

[9] Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., & Handley, M. (2011). Improving datacenter performance and robustness with multipath TCP. En *Computer Communication Review*, *41*(4), 266-277. [En línea]. doi: 10.1145/2043164.2018467. Consultado el 7 de mayo de 2023, desde https://inl.info.ucl.ac.be/system/files/mptcp-sigcomm.pdf.

[10] CORDIS. (2016). *MAMI Project — Measurement and Architecture for a Middleboxed Internet*. Consultado el 8 de mayo de 2023, desde https://cordis.europa.eu/project/id/688421

[11] Fossati, T., Müntener, R., & Neuhaus, S. (2018). *Security and Privacy Implications of Middlebox Cooperation Protocols*. [En línea]. Consultado el 8 de mayo de 2023, desde https://arxiv.org/ftp/arxiv/papers/1812/1812.05437.pdf

[12] *MAMI Project* [Implementación en Go de la versión 3 de la API de la *Path Transparency Observatory* (PTO) MAMI]. (2020). GitHub. Consultado el 8 de mayo de 2023, desde https://github.com/mami-project

[13] *5G MEDIA < 5G-PPP*. (2014, febrero). Consultado el 8 de mayo de 2023, desde https://5g-ppp.eu/5g-media/

[14] Coninck, Q. D. (2017a, junio). *Multipath QUIC*. Multipath QUIC. Consultado el 8 de mayo de 2023, desde https://multipath-quic.org/

[15] Coninck, Q. D. (2017b, septiembre). *A QUIC implementation in pure Go*. GitHub. Consultado el 8 de mayo de 2023, desde https://github.com/qdeconinck/mp-quic

[16] Postel, J. (1980, agosto). *User Datagram Protocol* (Request for Comments N.º 768). Internet Engineering Task Force. doi: 10.17487/RFC0768. Consultado el 8 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc768

[17] Iyengar, J., & Thomson, M. (2021, mayo). *QUIC: A UDP-Based Multiplexed and Secure Transport* (Request for Comments N.º 9000). Internet Engineering Task Force. doi: 10.17487/RFC9000. Consultado el 8 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc9000

[18] Bonaventure, O., & Coninck, Q. D. (2018). *Multipath QUIC: Design and Evaluation.* [Diapositivas de PowerPoint]. SlideShare. Consultado el 8 de mayo de 2023, desde https://slideplayer.com/slide/13752401/

[19] Coninck, Q. D. (2017c, junio). *MultipathTester*. Multipath QUIC. Consultado el 8 de mayo de 2023, desde https://multipath-quic.org/multipathtester.html

[20] Tessares. (2022, noviembre). *MPTCP App by Tessares SA*. AppAdvice. Consultado el 8 de mayo de 2023, desde https://appadvice.com/app/mptcp-app/1464506715

[21] Tessares. (2023, abril). *Seamless Multipath Connectivity Solutions from Tessares.* Consultado el 8 de mayo de 2023, desde https://www.tessares.net/

[22] Coudron, M., Secci, S., Pujolle, G., Raad, P., & Gallard, P. (2013). Cross-layer cooperation to boost multipath TCP performance in cloud networks. En *2013 IEEE 2nd International Conference on Cloud Networking (CloudNet)*, pp. 58-66. [En línea]. doi: 10.1109/CloudNet.2013.6710558. Consultado el 8 de mayo de 2023, desde http://ieeexplore.ieee.org/document/6710558/.

[23] Coudron, M., Secci, S., Pujolle, G., Raad, P., & Gallard, P. (2014, septiembre). *Cross-layer cooperation to boost multipath TCP performance in cloud networks.* [Diapositivas de PowePoint]. SlideServe. Consultado el 8 de mayo de 2023, desde https://www.slideserve.com/enid/cross-layer-cooperation-to-boost-multipath-tcp-performance-in-cloud-networks

[24] Vu, V. A., & Akselrod, M. (2021). An experiment of dual-LTE MPTCP with In-Car Voice Assistant. En *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, pp. 1-5. [En línea]. doi: 10.1109/VTC2021-Spring51267.2021.9448971. Consultado el 8 de mayo de 2023, desde https://ieeexplore.ieee.org/document/9448971.

[25] Chao, L., Wu, C., Yoshinaga, T., Bao, W., & Ji, Y. (2021). *A Brief Review of Multipath TCP for Vehicular Networks. Sensors*, *21*(8), p. 21. [Imagen]. Consultado el 8 de mayo de 2023, desde https://www.researchgate.net/publication/350897554_A_Brief_Review_of_Multipath_TCP_for_Vehicular_Networks.

[26] *Project - 5G-CLARITY Project.* (2019, agosto). 5G-CLARITY Project. Consultado el 8 de mayo de 2023, desde https://www.5gclarity.com/index.php/projects/

[27] Navarro Ortiz, J. (2023a, marzo). *Multi-technology testbed v0 setup* [Multi-connectivity testbed (virtualized)]. GitHub. Consultado el 8 de mayo de 2023, desde https://github.com/jorgenavarroortiz/multitechnology_testbed_v0

[28] WiMuNet - Wireless and Multimedia Networking Lab. (2019, noviembre). *5G-CLARITY project*. Consultado el 8 de mayo de 2023, desde https://wimunet.ugr.es/projects/5gclarity.php

[29] Bonaventure, O., Paasch, C., Raiciu, C., Detal, G., Barre, S., Ford, A., Handley, M., Wischik, D., & Li, Y. (2013a). *MultiPath TCP - Linux Kernel implementation : Home Page*. Consultado el 8 de mayo de 2023, desde https://www.multipath-tcp.org/

[30] *Linux MPTCP Upstream Project: Home*. (2023, junio). GitHub. Consultado el 8 de mayo de 2023, desde https://github.com/multipath-tcp/mptcp_net-next/wiki/Home

[31] NetScaler. (2023, mayo). *TCP Configurations.* TCP Configurations — NetScaler 13.1. Consultado el 8 de mayo de 2023, desde https://docs.netscaler.com/en-us/citrix-adc/current-release/system/tcp-configurations.html

[32] FreeBSD. (2020, julio). *Multipath TCP for FreeBSD.* Multipath TCP for FreeBSD — FreeBSD Foundation. Consultado el 8 de mayo de 2023, desde https://freebsdfoundation.org/project/multipath-tcp-for-freebsd/

[33] Oracle Solaris. (2013, enero). *Oracle Solaris SAN Configuration and Multipathing Guide*. Consultado el 8 de mayo de 2023, desde https://docs.oracle.com/cd/E26505_01/pdf/E35235.pdf

[34] Jonathan, P., & Shoch, J. (1981, septiembre). *Transmission Control Protocol* (Request for Comments N.º 793). Internet Engineering Task Force. doi: 10.17487/RFC0793. Consultado el 10 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc793

[35] Kale, C. J., & Socolofsky, T. J. (1991, enero). *TCP/IP tutorial* (Request for Comments N.º 1180). Internet Engineering Task Force. doi: 10.17487/RFC1180. Consultado el 10 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc1180

[36] Chiu, D.-M., & Jain, R. (1989). *Analysis of the Increase/Decrease Algorithms for Congestion Avoidance in Computer Netowrks.* [En línea]. Consultado el 10 de mayo de 2023, desde https://www.cse.wustl.edu/~jain/papers/ftp/cong_av.pdf

[37] Brakmo, L. S., & Peterson, L. L. (2012). *A Comparative Analysis of TCP Tahoe,Reno, New-Reno, SACK and Vegas*. [En línea]. Consultado el 10 de mayo de 2023, desde https://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf

[38] Montañana, R. (2019, mayo). *Curso de Redes Telemáticas - Videotutorial 10.18 Timer de persistencia.* Aula Clic. Consultado el 10 de mayo de 2023, desde https://www.aulaclic.es/redes/secuencias/p10_18_timer_persistencia.htm

[39] Pearce, C. (2014). *MPTCP Details added on to the end of a standard TCP packet*. [Imagen]. Multipath TCP, Pwning Today's Networks With Tomorrow's Protocols, p.3. Consultado el 10 de mayo de 2023, desde https://www.blackhat.com/docs/us-14/materials/us-14-Pearce-Multipath-TCP-Breaking-Todays-Networks-With-Tomorrows-Protocols-WP.pdf.

[40] Hmong.wiki. (s.f.). *TCP de múltiples rutas*. TCP de múltiples rutas BeneficiosyInterfaz de usuario. Consultado el 10 de mayo de 2023, desde https://hmong.es/wiki/Multipath_TCP

[41] Noh, G., Park, H., Roh, H., & Lee, W. (2019). *An example of MPTCP connection with two network path*. [Imagen]. Secure and Lightweight Subflow Establishment of Multipath-TCP. En *IEEE Access*, *7*, p. 177438. doi: 10.1109/ACCESS.2019.2957434. Consultado el 10 de mayo de 2023, desde https://www.researchgate.net/publication/337746690_Secure_and_Lightweight_Subflow_Establishment_of_Multipath-TCP.

[42] Krawczyk, H., Bellare, M., & Canetti, R. (1997, febrero). *HMAC: Keyed-Hashing for Message Authentication* (Request for Comments N.º 2104). Internet Engineering Task Force. doi: 10.17487/RFC2104. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc2104

[43] Eastlake 3rd, D. E., & Jones, P. (2001, septiembre). *US Secure Hash Algorithm 1 (SHA1)* (Request for Comments N.º 3174). Internet Engineering Task Force. doi: 10.17487/RFC3174. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc3174

[44] Hansen, T., & Eastlake 3rd, D. E. (2011, mayo). *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)* (Request for Comments N.º 6234). Internet Engineering Task Force. doi: 10.17487/RFC6234. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc6234

[45] Raiciu, C., Handley, M. J., & Wischik, D. (2011, octubre). *Coupled Congestion Control for Multipath Transport Protocols* (Request for Comments N.º 6356). Internet Engineering Task Force. doi: 10.17487/RFC6356. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc6356

[46] Khalili, R., Gast, N., Popovic, M., & Boudec, J.-Y. L. (2014, julio). *Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP* \ (Internet Draft draft-khalili-mptcp-congestion-control-00). Internet Engineering Task Force. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/draft-khalili-mptcp-congestion-control-00

[47] Imaduddin, M. F., Putrada, A. G., & Karimah, S. A. (2021). Multipath TCP Scheduling Performance Analysis and Congestion Control on Video Streaming on the MPTCP Network. En *2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM)*, pp. 562-567. [En línea]. doi: 10.1109/ICSECS52883.2021.00109. Consultado el 11 de mayo de 2023, desde https://ieeexplore.ieee.org/document/9536995

[48] Cao, Y., Xu, M., & Fu, X. (2012). Delay-based congestion control for multipath TCP. En *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1-10. [En línea]. doi: 10.1109/ICNP.2012.6459978. Consultado el 11 de mayo de 2023, desde https://ieeexplore.ieee.org/document/6459978

[49] Iyengar, J., Raiciu, C., Barre, S., Handley, M. J., & Ford, A. (2011, marzo). *Architectural Guidelines for Multipath TCP Development* (Request for Comments N.º 6182). Internet Engineering Task Force. doi: 10.17487/RFC6182. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc6182

[50] Egevang, K. B., & Srisuresh, P. (2001, enero). *Traditional IP Network Address Translator (Traditional NAT)* (Request for Comments N.º 3022). Internet Engineering Task Force. doi: 10.17487/RFC3022. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc3022

[51] Griner, J., Border, J., Kojo, M., Shelby, Z. D., & Montenegro, G. (2001, junio). *Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations* (Request for Comments N.º 3135). Internet Engineering Task Force. doi: 10.17487/RFC3135. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc3135

[52] Handley, M., Paxson, V., & Kreibich, C. (2001). Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. En *10th USENIX Security Symposium (USENIX Security 01)*, pp. 1-17. [En línea]. Consultado el 11 de mayo de 2023, desde https://www.usenix.org/legacy/events/sec01/full_papers/handley/handley.pdf

[53] Freed, N. (2000, octubre). *Behavior of and Requirements for Internet Firewalls* (Request for Comments N.º 2979). Internet Engineering Task Force. doi: 10.17487/RFC2979. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc2979

[54] Sotomonte, P., & Orlando, B. (2020). *Sistemas de detección de intrusiones: IDS vs sistemas de prevención de intrusiones: IPS*. [En línea]. Consultado el 11 de mayo de 2023, desde http://repository.unipiloto.edu.co/handle/20.500.12277/7427

[55] Bagnulo, M. (2011, marzo). *Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses* (Request for Comments N.º 6181). Internet Engineering Task Force. doi: 10.17487/RFC6181. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc6181

[56] Bagnulo, M., Paasch, C., Gont, F., Bonaventure, O., & Raiciu, C. (2015, julio). *Analysis of Residual Threats and Possible Fixes for Multipath TCP (MPTCP)* (Request for Comments N.º 7430). Internet Engineering Task Force. doi: 10.17487/RFC7430. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc7430

[57] Eddy, W. (2007, agosto). *TCP SYN Flooding Attacks and Common Mitigations* (Request for Comments N.º 4987). Internet Engineering Task Force. doi: 10.17487/RFC4987. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc4987

[58] Díez, J., Bagnulo, M., Valera, F., & Vidal, I. (2011). Security for multipath TCP: a constructive approach. En *International Journal of Internet Protocol Technology*, *6*(3), 146-155. [En línea]. doi: 10.1504/IJIPT.2011.043675. Consultado el 11 de mayo de 2023, desde https://www.researchgate.net/publication/220526807_Security_for_multipath_TCP_a_constructive_approach

[59] Aura, T. (2005, marzo). *Cryptographically Generated Addresses (CGA)* (Request for Comments N.º 3972). Internet Engineering Task Force. doi: 10.17487/RFC3972. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc3972

[60] Bittau, A., Giffin, D. B., Handley, M. J., Mazieres, D., Slack, Q., & Smith, E. W. (2019, mayo). *Cryptographic Protection of TCP Streams (tcpcrypt)* (Request for Comments N.º 8548). Internet Engineering Task Force. doi: 10.17487/RFC8548. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc8548

[61] Hoffman, P. E. (2023, febrero). *DNS Security Extensions (DNSSEC)* (Request for Comments N.º 9364). Internet Engineering Task Force. doi: 10.17487/RFC9364. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc9364

[62] Nordmark, E., & Bagnulo, M. (2009, junio). *Shim6: Level 3 Multihoming Shim Protocol for IPv6* (Request for Comments N.º 5533). Internet Engineering Task Force. doi: 10.17487/RFC5533. Consultado el 11 de mayo de 2023, desde https://datatracker.ietf.org/doc/rfc5533

[63] Martineau, M., & Baerts, M. (2019). *Multipath TCP Upstreaming.* [En línea]. Consultado el 11 de mayo de 2023, desde https://lpc.events/event/4/contributions/435/attachments/246/428/LPC2019-Upstreaming-MPTCP-paper.pdf

[64] *MPTCP Linux Testing* [Three main tools for test purposes: kselftest, syzkaller, and packetdrill]. (2023, abril). GitHub. Consultado el 11 de mayo de 2023, desde https://github.com/multipath-tcp/mptcp_net-next/wiki/Testing

[65] Watson, D. (2019). *KTLS: Linux Kernel Transport Layer Security.* [En línea]. Consultado el 11 de mayo de 2023, desde https://netdevconf.info/1.2/papers/ktls.pdf

[66] Oracle Corporation. (s.f.). *Oracle VM VirtualBox.* Consultado el 18 de mayo de 2023, desde https://www.virtualbox.org/

[67] HashiCorp. (s.f.). *Vagrant by HashiCorp.* Consultado el 18 de mayo de 2023, desde https://www.vagrantup.com/

[68] Mobatek. (s.f.). *MobaXterm free xserver and tabbed SSH client for windows.* Consultado el 18 de mayo de 2023, desde https://mobaxterm.mobatek.net/

[69] Grafana Labs. (s.f.). *Grafana: The open observability platform.* Consultado el 18 de mayo de 2023, desde https://grafana.com/

[70] MathWorks. (s.f.). *MATLAB Student.* Consultado el 18 de mayo de 2023, desde https://es.mathworks.com/products/matlab/student.html

[71] Wireshark Foundation. (s.f.). *Wireshark · Go Deep.* Consultado el 18 de mayo de 2023, desde https://www.wireshark.org/

[72] Microsoft Corporation. (s.f.). *Especificaciones y requisitos del sistema de Windows 11 — Microsoft.* Consultado el 18 de mayo de 2023, desde https://www.microsoft.com/es-es/windows/windows-11-specifications

[73] Ubuntu. (s.f.). *Ubuntu 18.04 LTS.* Consultado el 16 de septiembre de 2021, desde https://releases.ubuntu.com/18.04/?_ga=2.48740582.193470755.1688824759-946007595.1686128114

[74] Canonical Ltd. (s.f.). *Ubuntu 22.04 LTS.* Consultado el 18 de mayo de 2023, desde https://ubuntu.com/blog/tag/22-04-lts

[75] *Linux Kernel v5.x.* (s.f.). Consultado el 18 de mayo de 2023, desde https://cdn.kernel.org/pub/linux/kernel/v5.x/

[76] *Linux Kernel v6.x.* (s.f.). Consultado el 18 de mayo de 2023, desde https://cdn.kernel.org/pub/linux/kernel/v6.x/

[77] *multipath-tcp/mptcp-net_next* [Development version of the *Upstream MultiPath TCP* Linux kernel]. (2014, noviembre). GitHub. Consultado el 18 de mayo de 2023, desde https://github.com/multipath-tcp/mptcp_net-next

[78] Overleaf. (s.f.). *Overleaf — Editor LaTeX online.* Consultado el 18 de mayo de 2023, desde https://es.overleaf.com/project

[79] Thomas, A., & Barashev, D. (s.f.). *GanttProject: Free project management tool for windows, macOS and linux.* Consultado el 18 de mayo de 2023, desde https://www.ganttproject.biz

[80] *Center for History and New Media* (CHNM) — *George Mason University* (GMU). (s.f.). *Zotero: Your personal research assistant.* Consultado el 18 de mayo de 2023, desde https://www.zotero.org/

[81] Cech, H. (2020). *Schematic depiction of the Linux MPTCP components on the sending and the receiving sides.* [Imagen]. En Analyzing and Realizing Multipath TCP Schedulers in Linux, (p. 4). Consultado el 18 de mayo de 2023, desde https://www.nitindermohan.com/documents/student-thesis/HendrikCechGR.pdf.

[82] *multipath-tcp/mptcp* [Out-of-tree Linux Kernel implementation of MultiPath TCP]. (2023, mayo). GitHub. Consultado el 20 de mayo de 2023, desde https://github.com/multipath-tcp/mptcp

[83] Bonaventure, O., Paasch, C., Raiciu, C., Detal, G., Barre, S., Ford, A., Handley, M., Wischik, D., & Li, Y. (2023, mayo). *Index of /patches.* MultiPath TCP - Linux Kernel implementation. Consultado el 28 de mayo de 2023, desde http://multipath-tcp.org/patches/

[84] Bonaventure, O., Paasch, C., Raiciu, C., Detal, G., Barre, S., Ford, A., Handley, M., Wischik, D., & Li, Y. (2013b). *How to install MPTCP?.* MultiPath TCP - Linux Kernel implementation : Users - Do It Yourself. Consultado el 20 de mayo de 2023, desde https://multipath-tcp.org/pmwiki.php/Users/DoItYourself

[85] Bonaventure, O., Paasch, C., Raiciu, C., Detal, G., Barre, S., Ford, A., Handley, M., Wischik, D., & Li, Y. (2013c). *Configure rounting.* MultiPath TCP - Linux Kernel implementation : Users - Configure Routing. Consultado el 20 de mayo de 2023, desde https://multipath-tcp.org/pmwiki.php/Users/ConfigureRouting

[86] *Releases · multipath-tcp/mptcp* [Multipath TCP Linux Kernel Releases]. (2023, febrero). GitHub. Consultado el 20 de mayo de 2023, desde https://github.com/multipath-tcp/mptcp/releases

[87] Othman, O. (2023, mayo). *Multipath TCP Daemon.* GitHub. Consultado el 20 de mayo de 2023, desde https://github.com/multipath-tcp/mptcpd

[88] Oliver, A. (2023, 3 de junio). *MPTCP-v1* [Implementación de MPTCP v1 en un escenario virtual con VirtualBox]. GitHub. Consultado el 15 de junio de 2023, desde https://github.com/AlejandraOliver/MPTCP-v1/tree/main

[89] Ubuntu. (s.f.). *GitKernelBuild.* Ubuntu Wiki. Consultado el 10 de junio de 2023, desde https://wiki.ubuntu.com/KernelTeam/GitKernelBuild

[90] Yamada, M. (2023, 7 de junio). *Minimal requirements to compile the Kernel.* GitHub. Consultado el 10 de junio de 2023, desde https://github.com/multipath-tcp/mptcp_net-next/blob/export/Documentation/process/changes.rst

[91] Navarro Ortiz, J. (2022, 31 de marzo). *Kernel 5.5 with MPTCP support and WRR.* GitHub. Consultado el 10 de junio de 2023, desde https://github.com/jorgenavarroortiz/linux-kernel-5.5-mptcp-wrr

[92] *MPTCP Sysfs variables.* (2023, 16 de junio). GitHub. Consultado el 10 de junio de 2023, desde https://github.com/multipath-tcp/mptcp_net-next/blob/export/Documentation/networking/mptcp-sysctl.rst

[93] *Index of /pub/linux/utils/net/iproute2/*. (2023, 3 de mayo). Consultado el 10 de junio de 2023, desde https://mirrors.edge.kernel.org/pub/linux/utils/net/iproute2/

[94] Debian Webmaster. (s.f.). *Debian – Details of package ifstat in buster*. Consultado el 10 de junio de 2023, desde https://packages.debian.org/buster/ifstat

[95] Navarro Ortiz, J. (2023b, mayo). *Statistics for the multitechnology_testbed_v0 repo*. GitHub. Consultado el 10 de junio de 2023, desde https://github.com/jorgenavarroortiz/multitechnology_testbed_v0_stats

[96] Navarro Ortiz, J. (2023c, junio). *Testing WRR (without tunneling) with 5G NR + Wi-Fi 6 + LiFi*. GitHub. Consultado el 10 de junio de 2023, desde https://github.com/jorgenavarroortiz/multitechnology_testbed_v0_NUC

[97] *Multipath TCP* [Set of repositories with MPTCP tools]. (2023, mayo). GitHub. Consultado el 20 de mayo de 2023, desde https://github.com/orgs/multipath-tcp/repositories

Apéndice A

Archivos Necesarios en la Ejecución de Vagrant

Para que *Vagrant* pueda crear el escenario con MPTCP v0, se le proporcionan una serie de archivos con los que instalar y configurar las máquinas. Como se ha mencionado en el capítulo 5, se emplea el *Vagrantfile*, que a su vez utiliza una serie de ficheros más. El contenido se explica a continuación.

Vagrantfile

Este archivo se muestra en la figura A.4. Lo primero que se hace es especificar la versión de la API del archivo de configuración de *Vagrant*, y el número de máquinas virtuales que se van a crear a 2. Además, se configuran dichas máquinas con *Ubuntu* 18.04 LTS (la versión LTS más moderna es la 22.04, pero ésta se empleará en MPTCP v1 para que ambas implementaciones no sean con el mismo S.O y haya variedad) y el poder acceder a ellas mediante una conexión SSH con credenciales *vagrant:vagrant*. Seguidamente, se habilita el reenvío de agente SSH y X11 para permitir conexiones gráficas y reenvío de X11; además de deshabilitar las carpetas compartidas entre anfitrión y máquina virtual. Lo siguiente que lleva a cabo este archivo se realiza primero en una máquina, y luego en la otra:

- Configura el redireccionamiento de puertos para permitir la conexión SSH a cada máquina virtual. Ambas máquinas utilizan el puerto 22, el *host* utiliza el 12222 para la conexión con la máquina 1, y el 22222 para la conexión con la máquina 2.
- Asigna direcciones IP privadas dentro de la red interna *ue_ue* a las interfaces de red de cada máquina virtual. Estas IP se mostraron en la figura 5.2.
- Establece el nombre de la máquina en el entorno *Vagrant* (*mptcpUe1* y *mptcpUe2*).

- Configura opciones adicionales para *VirtualBox* como: permitir el modo promiscuo en las interfaces de red de *ue_ue*, establecer el nombre de la máquina en *VirtualBox* (el mismo que en *Vagrant*), habilitar la resolución de *Domain Name System* (DNS) en la máquina a través del anfitrión (este último actúa como *proxy* DNS), establecer la cantidad de RAM a 4096 MB, y 2 CPU virtuales.

- Una vez la máquina se ha creado, dentro de ella se ejecutan una serie de *scripts*. Además, se utiliza el comando $*reload* para hacer una recarga de la máquina, en caso de que alguna tarea que se lleve a cabo en dichos *scripts* lo requiera. Estos ejecutables se describen seguidamente:

mptcp_kernel54144_installation.sh

Se muestra en la figura A.1 y en él se realizan una serie de tareas:

- Se actualiza la lista de paquetes disponibles en el sistema operativo, y se instalan las dependencias necesarias para poder compilar e instalar un nuevo *kernel* en la máquina (el *kernel* predeterminado es el 4.15).

- Se instalan los paquetes relacionados con el *kernel* 5.4, ya con MPTCP v0 incorporado. Como se mencionó en el capítulo 3, el último *kernel* con soporte a la versión 0 del protocolo es el 5.5, pero como en los paquetes relacionados con la versión 5.4 el parche MPTCP v0 incluye uno de los tipos de *schedulers* en su versión más moderna, se escoge éste en caso de que se desee probar (se incluye el *weightly round-robin scheduler*, el cual funciona como el *round-robin* pero añadiendo pesos en las interfaces).

- Por último, se instala *openvpn* en caso de que se desee añadir alguna *Virtual Private Network* (VPN) al escenario.

```
1  #!/bin/bash
2
3  sudo apt-get -y update
4  sudo apt-get -y install build-essential libncurses-dev bison flex
5
6  cd $HOME/vagrant/MPTCP0.96_kernel5.4.144_WRR05
7  sudo dpkg -i linux-image-5.4.144-mptcp-wrr05-5gclarity+_5.4.144-mptcp-
       wrr05-5gclarity+-2_amd64.deb
8  sudo dpkg -i linux-headers-5.4.144-mptcp-wrr05-5gclarity+_5.4.144-
       mptcp-wrr05-5gclarity+-2_amd64.deb
9  sudo dpkg -i linux-libc-dev_5.4.144-mptcp-wrr05-5gclarity+-2_amd64.deb
10
11 sudo apt-get -y install openvpn
```

Figura A.1: Instalación del *Kernel* 5.4.144 y de *Openvpn*.

mptcp_installation.sh

En este *script* (véase la figura A.2) se instalan varias herramientas de línea de comandos necesarias para el manejo y control del protocolo en las máquinas. Éstas se alojan en una serie de repositorios de GitHub que se pueden encontrar en [97]. Su compilación e instalación se lleva a cabo mediante los comandos *make* y *make install*, estos comandos se utilizan en Linux para compilación e instalación de software. Las herramientas son las siguientes:

- ***Git*** y ***wget***: estas herramientas se usan para el control de versiones, clonación de repositorios, descarga de archivos de Internet, etc.
- **Colección de programas que forman el conjunto base de la "NET-3"para el sistema operativo Linux**: entre las herramientas instaladas se encuentran *arp*, *hostname*, *ifconfig*, *netstat*, *rarp* y *route*.
- ***mptcptrace***: estudia las conexiones MPTCP a partir de un fichero *.pcap*.
- ***iproute-mptcp***: extensión de *iproute2* con soporte MPTCP. Permite la configuración de políticas de enrutamiento, visualización de conexiones activas, control de rutas e interfaces, todo ello de MPTCP.
- Para finalizar, se instalan varias herramientas:
  - ***bridge-utils***: utilidades de puente de red.
  - ***xplot.org***: visualización de datos en forma de gráficos.
  - ***gnuplot***: generación de estadísticas y gráficos.
  - ***iperf***: supervisión del rendimiento de la red.
  - ***ifstat***: control y monitorización de estadísticas de interfaz de red.
  - ***wireless-tools*** : gestión de redes inalámbricas.
  - ***tshark***: captura de paquetes y análisis de protocolos de red.

```
sudo apt-get -y install git wget

# Net-tools supporting MPTCP (e.g. netstat -m)
cd $HOME
git clone https://github.com/multipath-tcp/net-tools.git
cd net-tools
cp $HOME/vagrant/net-tools/config.h .
make
sudo make install

# MPTCPtrace
sudo apt-get -y install check libssl-dev libpcap-dev autoconf pkg-config
cd $HOME
git clone https://github.com/multipath-tcp/mptcptrace
cd mptcptrace
./autogen.sh
./configure
make
sudo make install

# iproute-mptcp (required for command "ip link set dev <interface>
    multipath <off/on/backup>")
cd $HOME
git clone https://github.com/multipath-tcp/iproute-mptcp.git
cd iproute-mptcp
make
sudo make install

# Tools
cd $HOME
sudo apt-get -y install bridge-utils xplot.org gnuplot iperf ifstat
    wireless-tools
sudo bash -c 'DEBIAN_FRONTEND=noninteractive apt-get -y install tshark'
```

Figura A.2: Instalación de Herramientas para MPTCP v0.

initial_testing_2machines.sh

En este último archivo, mostrado en la figura A.3, se clona el repositorio utilizado en este proceso (*multitechnology_testbed_v0*) dentro de las máquinas, ya que además de los archivos *vagrant*, contiene otros que se utilizarán más tarde para la comprobación del funcionamiento y realización de pruebas con MPTCP v0. Después se renombra el repositorio a *free5gc*.

```
# INITIAL TESTING (for VMs not installing the repository)
cd $HOME
git clone https://github.com/jorgenavarroortiz/multitechnology_testbed_v0.git
mv multitechnology_testbed_v0 free5gc
```

Figura A.3: Clonación de *multitechnology_testbed_v0* en las Máquinas.

Por último, se presenta el *Vagrantfile*:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

# Vagrantfile API/syntax version. Don't touch unless you know what you're doing!
VAGRANTFILE_API_VERSION = "2"
#Vagrant::DEFAULT_SERVER_URL.replace('https://vagrantcloud.com')

# Number of virtual machines (configured as linked clones to save harddisk)
VMS_COUNT = 2
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|

  # Every Vagrant virtual environment requires a box to build off of.
  config.vm.box = "bento/ubuntu-18.04"
  config.ssh.username = 'vagrant'
  config.ssh.password = 'vagrant'
  config.ssh.insert_key = false

  # Custom: SSH graphical, X11
  config.ssh.forward_agent = true
  config.ssh.forward_x11 = true

  # avoid mounting shared folder
  config.vm.synced_folder '.', '/vagrant', disabled: true

  ####################
  # Defining the MPTCP UE i VM
  ####################
  (1..VMS_COUNT).each do |i|
    config.vm.define "mptcpUe#{i}" do |mptcpUe|
      mptcpUe.vm.network "forwarded_port", guest: 22, host: "#{i*10000+2222}",
          protocol: "tcp"                        # Port forwarding through the NAT
          interface
      mptcpUe.vm.network "private_network", ip: "10.1.1.#{3*(i-1)+1}", auto_config:
          true, virtualbox__intnet: "ue_ue" # Interface 1 connecting to second machine
      mptcpUe.vm.network "private_network", ip: "10.1.1.#{3*(i-1)+2}", auto_config:
          true, virtualbox__intnet: "ue_ue" # Interface 2 connecting to second machine
      mptcpUe.vm.network "private_network", ip: "10.1.1.#{3*(i-1)+3}", auto_config:
          true, virtualbox__intnet: "ue_ue" # Interface 3 connecting to second machine
      mptcpUe.vm.hostname = "mptcpUe#{i}"
      mptcpUe.vm.provider :virtualbox do |vm|
        # Linked clones
        #vm.linked_clone = true
        # Configure networking interfaces
        vm.customize ["modifyvm", :id, "--nicpromisc2", "allow-all"]
        vm.customize ["modifyvm", :id, "--nicpromisc3", "allow-all"]
        vm.customize ["modifyvm", :id, "--nicpromisc4", "allow-all"]
        # Config name that appears in virtual box
        vm.name = "mptcpUe#{i}"
        # DNS queries to the host, which becomes a DNS Proxy
        vm.customize ["modifyvm", :id, "--natdnshostresolver1", "on"]
        # Lease more RAM to the guest
        vm.customize ["modifyvm", :id, "--memory", "4096"]
        # Set number of CPUs
        vm.customize ["modifyvm", :id, "--cpus", 2]
      end
      # Provisioning
      mptcpUe.vm.provision "file", source: "./vagrant", destination: "$HOME/vagrant"
      mptcpUe.vm.provision "shell", path: "vagrant/mptcp_kernel54144_installation.sh",
           privileged: false
      mptcpUe.vm.provision :reload
      mptcpUe.vm.provision "shell", path: "vagrant/mptcp_installation.sh", privileged:
           false
      mptcpUe.vm.provision "shell", path: "vagrant/initial_testing_2machines.sh",
          privileged: false
    end
  end
end
```

Figura A.4: Archivo de Configuración *Vagrantfile* utilizado por *Vagrant*

Apéndice B

Configuración del *Routing* en el Escenario con MPTCP Versión 1

En este apéndice se muestran los *scripts* empleados para la configuración del *routing* en el escenario virtual con MPTCP versión 1.

Siguiendo los pasos de [85], se ejecutan los comandos de la figura B.1 en el cliente:

```
#!/bin/bash

sudo ip rule add from 10.1.1.1 table 10
sudo ip rule add from 10.1.1.2 table 20
sudo ip rule add from 10.1.1.3 table 30

sudo ip route add 10.1.1.0/24 dev enp0s8 scope link table 10
sudo ip route add default via 10.1.1.4 dev enp0s8 table 10

sudo ip route add 10.1.1.0/24 dev enp0s9 scope link table 20
sudo ip route add default via 10.1.1.4 dev enp0s9 table 20

sudo ip route add 10.1.1.0/24 dev enp0s10 scope link table 30
sudo ip route add default via 10.1.1.4 dev enp0s10 table 30

sudo ip route add default scope global nexthop via 10.1.1.4 dev enp0s8
sudo ip route del 169.254.0.0/16
```

Figura B.1: *Routing* en el Cliente.

Como se mostró en la sección 5.2.3, primero se crean 3 tablas de enrutamiento basadas en la IP de origen, seguidamente se configura cada una mediante dos comandos: el primero indica que cualquier tráfico destinado a la red con el prefijo 10.1.1.0/24, debe enviarse directamente a través del

dispositivo de red enp0s8, enp0s9 o enp0s10 (dependiendo de la tabla que se esté configurando). El segundo indica que para el tráfico que salga de la interfaz enp0s8 (tabla 1), enp0s9 (tabla 2) o enp0s10 (tabla 3), y que no tenga destino definido, la IP por defecto será la IP de la interfaz enp0s8 del otro extremo (10.1.1.4).
En cuanto a los dos últimos comandos, el primero establece la ruta por defecto para el tráfico de Internet (lo que se ha dicho que es NAT), y el segundo elimina un enlace local que se crea por defecto y que no es necesario.

Para el caso del servidor, mostrado en la figura B.2, los comandos a utilizar son los mismos, cambiando la IP de las interfaces a la que corresponde ahora.

```
#!/bin/bash

sudo ip rule add from 10.1.1.4 table 10
sudo ip rule add from 10.1.1.5 table 20
sudo ip rule add from 10.1.1.6 table 30

sudo ip route add 10.1.1.0/24 dev enp0s8 scope link table 10
sudo ip route add default via 10.1.1.1 dev enp0s8 table 10

sudo ip route add 10.1.1.0/24 dev enp0s9 scope link table 20
sudo ip route add default via 10.1.1.1 dev enp0s9 table 20

sudo ip route add 10.1.1.0/24 dev enp0s10 scope link table 30
sudo ip route add default via 10.1.1.1 dev enp0s10 table 30

sudo ip route add default scope global nexthop via 10.1.1.1 dev enp0s8
sudo ip route del 169.254.0.0/16
```

Figura B.2: *Routing* en el Servidor.

Apéndice C

Script de *MatLab* para la Obtención de la CDF de los *Schedulers*

En este apéndice se muestra el *script* de *MatLab* empleado para el procesamiento de los datos extraídos de las diferentes pruebas con *schedulers*, con el fin de obtener una gráfica con las CDF de cada uno. Este *script* se observa en las figuras C.1 y C.2:

```
figure;
hold on;
xlimmax=300;

filename="result_default_conTC_processed.txt"
T = readtable(filename);
datos=T{:,1}/1e6;
h(1,1) = cdfplot(datos);

filename="result_redundant_conTC_processed.txt"
T = readtable(filename);
datos=T{:,1}/1e6;
h(1,2) = cdfplot(datos);

filename="result_roundrobin_conTCyCY_processed.txt"
T = readtable(filename);
datos=T{:,1}/1e6;
h(1,3) = cdfplot(datos);

filename="result_roundrobin_conTCyCN_processed.txt"
T = readtable(filename);
datos=T{:,1}/1e6;
h(1,4) = cdfplot(datos);
```

Figura C.1: Obtención de la CDF de cada *Scheduler* (Parte 1).

```
% https://www.mathworks.com/help/matlab/creating_plots/specify-plot-
    colors.html
set( h(:,1), 'LineStyle', '-', 'Color', [0 0.4470 0.7410]);
set( h(:,2), 'LineStyle', '-', 'Color', [0.4660 0.6740 0.1880]);
set( h(:,3), 'LineStyle', '-', 'Color', [0.8500 0.3250 0.0980]);
set( h(:,4), 'LineStyle', '-', 'Color', [0.4940 0.1840 0.5560]);


title('Throughput of MPTCP v0 schedulers');
xlabel('throughput (Mbps)');
ylabel('CDF');
legend({'Default', 'Redundant', 'Round-Robin C=Y', 'Round-Robin C=N'},
    'FontSize',12);
ax = gca;
ax.FontSize = 12;


saveas(gcf, "iperf_mptcpv0_all.fig");
saveas(gcf, "iperf_mptcpv0_all.png");
```

Figura C.2: Obtención de la CDF de cada *Scheduler* (Parte 2).

El código hace lo siguiente:

1. Se crea una figura que va a albergar múltiples gráficos en ella.

2. Se establece el límite máximo del eje X en 300, ya que en este eje va a ir dibujado el *throughput* y este es valor máximo que puede alcanzar en cualquiera de los casos.

3. Para cada archivo:

   - Se lee en forma de tabla. De esta manera, como cada uno contiene una sola columna de valores de *throughput*, se extrae dicha columna y se divide entre $1x10^6$ para pasar los valores de bits por segundo (bps) a megabits por segundo (Mbps). Estos nuevos valores se guardan en la variable *datos*.
   - Se utiliza la función *cdfplot* para trazar la función de distribución acumulada (CDF) de los valores de la variable *datos* en cada caso. El gráfico resultante se almacena en la variable h(1,1...4).

4. Se emplea la función *set* para establecer el estilo que van a tener las líneas y gráficos generados, es decir, los h(1,:). Los códigos de color se pueden obtener del enlace proporcionado al principio de la figura C.2.

5. Se establece: el título de la gráfica, el eje X que mostrará los valores de *throughput* en Mbps, el eje Y que mostrará la CDF, y la leyenda que corresponde. Además, se accede al objeto de ejes mediante la función *gca*, y se asigna a la variable *ax*. A continuación, se utiliza *ax* para

establecer el tamaño de la fuente de los ejes en 12 puntos, con el fin de que se vean mejor.

6. Se guarda la figura resultante en dos formatos: ".fig" y ".png", utilizando los nombres de archivo *iperf_mptcpv0_all.fig* y *iperf_mptcpv0_all.png*, respectivamente.

Como se ha podido analizar en las figuras C.1 y C.2, el *script* elabora una gráfica con los resultados de los *schedulers* de MPTCP v0. Para el caso de MPTCP v1, simplemente habría que cambiar los archivos de texto que contienen el *throughput*, por los que correspondan en este caso. Además, se deberá cambiar tanto el título y leyenda de la gráfica, como el nombre de los archivos ".fig" y ".png" que se almacenan. Todo lo demás se mantendría igual.