ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

*Proyecto Fin de Carrera*

# Análisis y mejora del comportamiento del protocolo MPTCP sobre escenarios de red heterogéneos LTE/WiFi

## (Analysis and enhancement of the MPTCP behavior over LTE/WiFi heterogeneous network scenarios)

Para acceder al Titulo de

**INGENIERO DE TELECOMUNICACIÓN**

Autor: Iván Fontán González

Julio - 2016

# Índice de contenidos

# Índice de figuras

# Índice de tablas

# Lista de acrónimos

**MPTCP** Multipath TCP

**IETF** Internet Engineering Task Force

**TCP** Transmission Control Protocol

**IP** Internet Protocol

**LTE** Long Term Evolution

**ns-3** Network Simulator 3

**MPC** Multipath Capable

**DSN** Data Sequence Number

**ACK** Acknowledgement

**RTT** Roundtrip time

**SACK** Selective ACK

**UE** User Equipment

**PDN** Packet Data Network

**UMTS** Universal Mobile Telecommunications System

**E-UTRAN** Evolved UTRAN

**SAE** System Architecture Evolution

**EPC** Evolved Packet Core

**EPS** Evolved Packet System

**QoS** Quality of Service

**P-GW** PDN Gateway

**S-GW** Serving Gateway

**MME** Mobility Management Entity

**PCRF** Policy Control and Charging Rules Function

**HSS** Home Subscriber Server

**IBSS** Independent Basic Service Set

**BSS** Basic Service Set

**ESS** Extended Service Set

**PBSS** Personal BSS

**PMD** Physical Medium Dependent

**PLCP** Physical Layer Convergence Procedure

**CSMA** Carrier Sense Multiple Access

**CA** Collision Avoidance

**Tcl** Tool Command Language

**OTcl** Object TCL

# Resumen ejecutivo

El actual estado de la tecnología, los múltiples accesos disponibles para conectarse a la red combinado con el sobredimensionado de redes que ofrece los proveedores de internet proponen un replanteamiento del esquema de conexión clásico en el que un terminal emplea una de sus interfaces, y solamente una, a pesar de disponer de varias, para acceder al *host* remoto y establecer su comunicación. La respuesta a este problema se da en forma de tecnologías multi-camino. Tecnologías capaces de aprovechar los diversos caminos o *paths* que ofrecen las redes de hoy en día. Esta solución permite un mayor rendimiento, y una mayor robustez frente a situaciones concretas.

El presente documento realiza un estudio de las posibilidades que puede otorgar el empleo de las tecnologías multi-camino, en concreto el protocolo MPTCP sobre redes inalámbricas y redes combinadas (LTE-Wifi). EL protocolo MPTCP es una extensión del protocolo TCP que da cobertura a una conexión multi-camino en la que se establecen varias conexiones TCP recogidas sobre una conexión MPTCP global.

Para realizar el estudio, el trabajo se apoya en simulaciones realizadas gracias a la herramienta ns-3 (Network Simulator 3), un simulador altamente flexible que permite la implementación de tecnologías sobre su núcleo para poder simularlo cualquier desarrollo.

# Executive Abstract

Nowadays, networks offer multiples access, moreover, operators use to duplicate links and equipment to get a higher resilience and protect the network. This invites us to change the classic communications paradigm in which a terminal connects to the remote host using just one interface. The multipath techniques might offer a suitable solution. A technology able to take advantage of various network paths offering higher performance and resilience than a conventional connection.

This document studies the advantages of using multipath techniques (specifically MPTCP) in Wireless networks. MPTPC protocol is an extension of the TCP protocol. A MPTCP connection is formed by different TCP connections.

We have used the ns-3 (Network Simulator) framework to obtain the several simulations of the scenarios. ns-3 is a highly flexible tool that incorporates many technologies.

# 1

# Introducción

En este capítulo introductorio se describe y analizan los motivos por los que se ha desarrollado este trabajo. Junto a ello, se presentan los objetivos que busca cumplir la memoria y por último se presenta la estructura y organización del documento, buscando una primera aproximación a la temática que tratará.

## 1.1 - Descripción

En la actualidad el gran desarrollo de las tecnologías disponibles, así como los grandes despliegues de red que llevan a cabo los proveedores son una realidad. A nadie le resulta extraño tener en el bolsillo un dispositivo móvil que esté conectado a la red de datos, y que a la vez tenga habilitada la conexión a la red wifi allá donde se encuentre. Todo esto lleva a replantear el concepto clásico de conexión en el que un dispositivo se conecta a una red a través de una única interfaz. ¿Sería posible, ya que el terminal tiene múltiples posibilidades de acceso a la red, usar más de una al mismo tiempo con el propósito de aumentar el rendimiento de la red? O quizás, si una de las redes a las que está conectado el dispositivo móvil no está disponible, por la razón que fuese ¿podría el dispositivo reaccionar a esta situación empleando otra de las redes a las que está conectado y evitando aquella que es problemática?

La respuesta que dio la comunidad investigadora a este problema fue el protocolo MPTCP, como resultado de las labores de un grupo de trabajo del IETF. MPTCP no es más que una extensión del protocolo TCP de internet que consiste en establecer múltiples conexiones simultáneamente cuyo objetivo es evitar los posibles fallos que presente la red. Gracias a MPTCP las conexiones pueden optar a un rendimiento más alto, además de una mayor estabilidad ante dichos fallos.

Los estudios realizados muestran las ventajas que posee el protocolo MPTCP frente a diferentes escenarios y situaciones determinadas. Uno de los intereses en este protocolo reside en los algoritmos empleados a la hora de determinar cuál de los múltiples caminos disponibles elegir

para enviar los datos. Las posibilidades van desde elegir uno de ellos de forma aleatoria, hasta complejos algoritmos que calculan parámetros en cada instante buscando tomar la decisión optima basándose en el estado de las redes, su velocidad de respuesta o su estado de congestión.

Por último, y desde un punto de vista más "terrenal" la implementación del protocolo en un simulador suena interesante. Una implementación dotada de las mayores funcionalidades disponibles. Aunque cada vez los desarrollos son más completos, aún hay funciones de las implementaciones que no producen los resultados que deberían o directamente otras que no están implementadas. Completar el desarrollo de un módulo MPTCP completo y preparado para las simulaciones es una de las direcciones por las que caminan los investigadores.

## 1.2 – Objetivos del trabajo

Es cierto que de los numerosos estudios que existen sobre esta tecnología, la gran mayoría se centran en el comportamiento del protocolo MPTCP en situaciones normales y son pocos los que se enfrentan a medios inalámbricos, como Wifi o LTE. Como también son pocos los que se centran en el estudio de esa toma de decisión respecto a que camino o flujo elegir para enviar los datos. Para el desarrollo de las simulaciones, la experimentación y observación se empleará el simulador de redes ns-3 (Network Simulator-3). Se trata de un simulador de código abierto basado en módulos, ofreciendo así una gran adaptabilidad a las necesidades del usuario a la hora de implementar nuevos módulos que permitan la simulación de nuevas tecnologías. Además, cuenta con una gran comunidad que respalda nuevos desarrollos y facilita el acercamiento de nuevos usuarios.

Actualmente, no existe ningún modulo "oficial" de MPTCP para ns-3, por lo que en el desarrollo de este trabajo se empleara un módulo en desarrollo de dicho protocolo que se rige por la especificación del IETF, especificación que se detallara en capítulos posteriores. En un origen el modulo con el que se inició este trabajo fue uno desarrollado para la versión 6 del simulador, que fue portado a la versión 13. Una primera parte del trabajo consistía en portar esta versión a la versión 17. Posteriormente apareció un módulo [16] para la versión 21, y se optó por cambiar a dicho modulo, ya que posee más prestaciones implementadas que el anterior, produce menos problemas y posee una estructura más sólida, estable y clara.

El objetivo del trabajo consiste, por un lado en adaptar el código del módulo para futuros proyectos y comprender su funcionamiento detenidamente; por otro lado, observar y estudiar el comportamiento de MPTCP en distintos escenarios Wireless, en concreto, como se comporta MPTCP frente a un escenario en el que dispone de dos tecnologías como LTE y 802.11 para acceder al destino, y comprobar el rendimiento de los distintos algoritmos de control de congestión y *scheduling* que se han propuesto últimamente.

## 1.3 – Estructura de la memoria

Esta sección despieza la estructura de la memoria, de forma que en unas breves líneas se pueda conocer el contenido y la dirección a la que apuntan cada uno de los capítulos:

- Capítulo 2 – Estado del arte. Capitulo que pretende dar a conocer al lector las nociones teóricas básicas sobre las que se apoya el desarrollo del trabajo. En concreto, se verá la arquitectura MPTCP, sus funcionalidades, problemas que presenta y como se dan solución a los mismos, procesos que yacen bajo una ejecución de una comunicación basada en el protocolo. También se verán de forma detallada los algoritmos de control de congestión y de *scheduling* que se proponen en la actualidad. Por último, y ya que parte del desarrollo del trabajo se basa en redes wifi y LTE, se dará al lector unas nociones básicas, sin entrar en profundidad, del funcionamiento de estas dos tecnologías.

- Capítulo 3 – Desarrollo en ns-3. Este capítulo abarca todo el desarrollo que se ha realizado sobre el simulador de redes, dando una introducción y descripción del mismo. Se explorarán los diferentes módulos y modelos que han tenido que estudiarse para el desarrollo del trabajo, y se expondrán las adaptaciones realizadas al código fuente del modelo MPTCP con el fin de adaptarlo a las necesidades planteadas en la sección 1.2. Se detallará cómo funciona la comunicación MPTCP a través de este módulo, entrando al código y viendo a que funciones se llama o que variables modifica. Se terminará con las implementaciones realizadas de los algoritmos de control de congestión y *scheduling* propuestos.

- Capítulo 4 – Resultados obtenidos. Este capítulo recoge todos los resultados obtenidos de las simulaciones que buscan saciar los objetivos planteados anteriormente. En un primer lugar se expondrá el proceso que se ha seguido para obtener los resultados, siendo necesario, en ocasiones, entrar al código fuente y manipularlo para obtener los datos deseados. Se explicarán mediante esquemas y descripciones los escenarios planteados para el estudio del trabajo, dando sus características, problemas que surgieron o problemas que pueden presentarse. La última sección del capítulo muestra los resultados obtenidos en forma de valores numéricos y gráficas, explicándolos y describiéndolos.

- Capítulo 5 – Conclusión. Como capítulo final se da una breve y vuelta a todo el camino recorrido para presentarlo como resultado final. Se da la conclusión a la que se llega tras el análisis del capítulo cuarto. Para terminar, se dejan abiertas varias líneas de investigación que han surgido con el desarrollo del trabajo que pueden dar lugares a futuros proyectos de alumnos.

# 2

# Estado del arte

Este capítulo introduce la base teórica sobre la cual se desarrolla todo el trabajo, dando así los conocimientos básicos al lector para que este pueda comprender los diferentes temas que se trataran en los capítulos posteriores.

## 2.1 - MPTCP

Ordenadores con una conexión Ethernet y una interfaz de conexión inalámbrica, *Smartphones* y *Tablets* que pueden conectarse a internet vía Wifi o por redes celulares (3G o 4G). En la actualidad los dispositivos disponen más de una interfaz de conexión. A esto le añadimos que los operadores de red, normalmente, duplican enlaces y equipo con el fin de proteger las redes contra fallos, especialmente en el acceso y en los enlaces intermedios. Junto a esto, tenemos el hecho de que la red dorsal (*backbone*) es una red mallada, generalmente. Con todo esto en mente, se puede apreciar que existen múltiples caminos entre dos nodos finales (*endpoints*) diferentes y surge la idea de usar varios caminos (*paths*) simultáneamente para mejorar la robustez y rendimiento de las conexiones. Estas conexiones con varios caminos (*multipath*) pueden balancear la cargar entre los diferentes *paths*, dinámicamente buscando evitar congestiones o los enlaces rotos.

Muchos estudios [1], [2], [3], [4], [5], [6], [7] han considerado la implementación del *multipath* en diferentes capas: en la capa de aplicación, transporte, etc. Siendo la capa de transporte la que parece mejor a la hora de implementar las capacidades *multipath*. En dicha capa, los sistemas pueden acopiar información sobre cada *path*, tales como: capacidad, latencia, estado de congestión. Esta información puede ser usada para reaccionar ante eventos de congestión en la red, evitando así el tramo o enlaces congestionados. Para desarrollar un protocolo *multipath* en la capa de transporte se creó un grupo (*working group*) del IETF, MultiPath TCP

(MPTCP). Este grupo desarrolla un protocolo basado en TCP (el protocolo más usado en la capa de transporte en Internet) para gestionar y manipular diferentes sub-conexiones, que usen diferentes *paths* en una conexión entre dos *endpoints*.

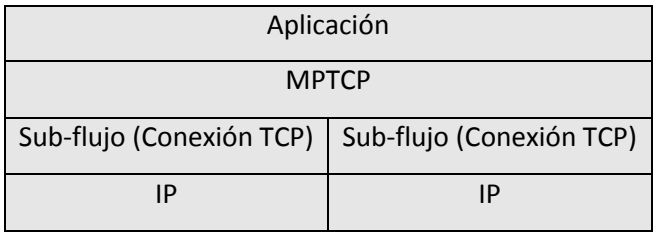


*Figura 2.1 – Arquitectura MPTCP*

El *working group* propuso MPTCP [8] (una extensión de TCP para capaz de gestionar múltiples caminos entre los dos nodos finales) para la capa de transporte. El protocolo MPTCP fue diseñado con tres objetivos claros:

1. **Mejorar el *Throughput***: El rendimiento del uso de varios flujos por diferentes caminos debe ser como mínimo igual al rendimiento al usar un único flujo que usa el mejor *path*.
2. **No dañar la red**: El empleo de varios caminos no debe usar más capacidades que los que se emplearían en cada uno de forma individual.
3. **Equilibrar la congestión**: el flujo debe ser capaz de desviar todo el tráfico como le sea posible para evitar la congestión.

El protocolo, de forma resumida, permite la conexión entre dos nodos finales, donde cada uno deberá de disponer de más de una dirección IP (varias interfaces). Una vez establecida la conexión, se buscarán múltiples flujos de datos (*flows*) y tanto el transmisor o el receptor, podrá añadirlos, eliminarlos y gestionarlos. Permitiendo priorizar uno de ellos sobre los demás, repartir la carga en todos ellos o usar alguna estrategia en concreto a la hora de seleccionar que flujos se usaran. También permite el uso de diversas tecnologías de nivel físico, es decir, permite usar diferentes redes para llegar al destino.

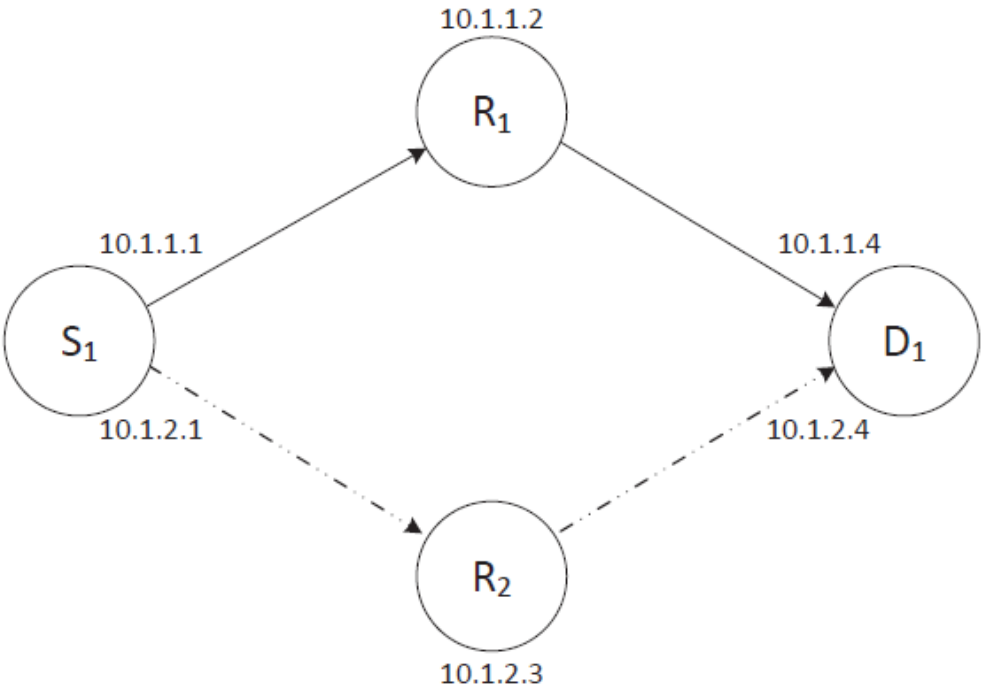


*Figura 2.2 – Esquema de una topología sencilla MPTCP*

La figura 2.2 muestra un esquema del funcionamiento básico de MPTCP. La topología presentada es la más usada a la hora de estudiar los beneficios del protocolo. Se puede apreciar en línea continua el *path* que usaría el TCP tradicional frente a los dos flujos que usaría el MPTCP.

### 2.1.1 - Funcionamiento de MPTCP

La capa de transporte en MPTCP está dividida en dos sub-capas, donde, la superior se encarga de las funcionalidades para gestionar la conexión, establecer la conexión, reordenar los paquetes… La inferior gestiona una serie de sub-flujos. Cada sub-flujo puede verse por sí solo como un flujo TCP único. MPTCP mantiene dos números de secuencia independientes, uno para cada sub-capa. Cada sub-flujo tiene su propia secuenciación, de manera similar al número de secuencia que vemos en el TCP estándar, identificando bytes dentro del sub-flujo. A nivel de conexión se emplea otra secuenciación, con el fin de encargarse del reordenado de segmentos TCP antes de que se manden a la capa de aplicación.

El protocolo MPTCP emplea opciones para intercambiar la información de señalización entre parejas, especialmente:

> **MPC** (Multipath Capable) se emplea durante el saludo a tres vías (*three-way handshake*) para establecer una conexión TCP *multipath*.
>
> **DATA FIN** se usa en para informar a la receptor remoto del fin de los datos además de decirla al nodo remoto que debe cerrar la conexión.

**ADD** y **REMOVE** se emplean para informar al nodo remoto de la disponibilidad de una nueva dirección (IPv4) o para ignorar una que ya existe.

**JOIN** cuya función es la de iniciar un nuevo sub-flujo entre dos direcciones a las que aún no se han dado uso, es decir, no hay establecido ningún flujo entre ellas.

**DSN** (*Data Sequence Number*) es un mapeado entre el nivel de sub-flujo y el espacio de números de secuenciación.

#### 2.1.1.1 - Establecimiento de la conexión

El proceso de establecer conexión es similar al que se puede ver en TCP (*three way handshake*) añadiendo el empleo de la opción MPC para informar al otro extremo que el lado que ha iniciado la conexión está dispuesto a enviar datos usando *Multipath* TCP.

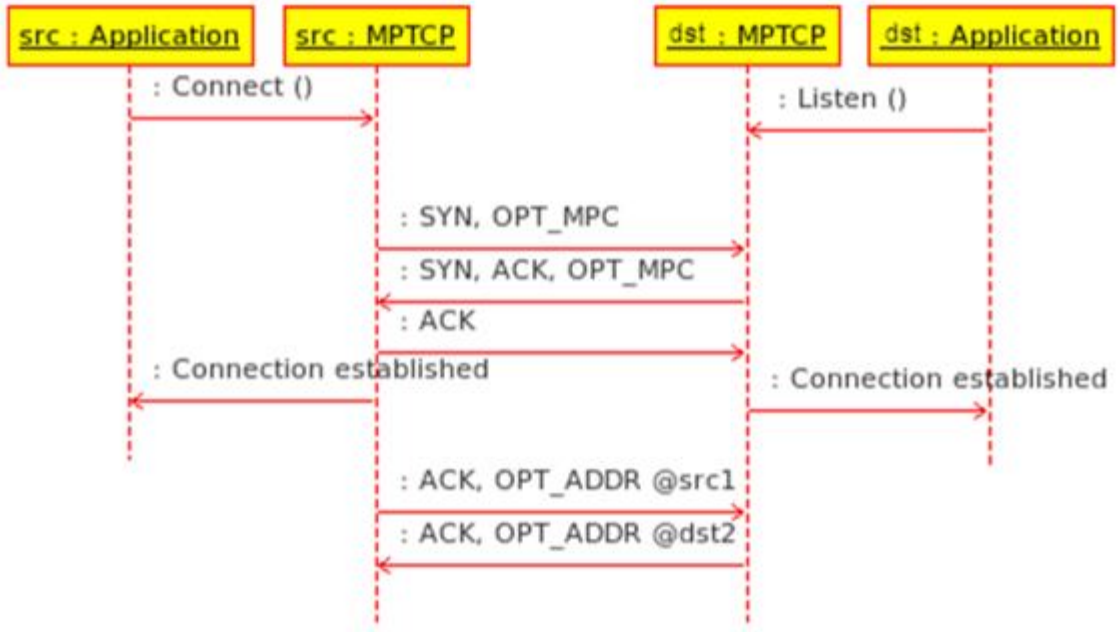


*Figura 2.3 – Establecimiento de conexión MPTCP*

La figura 2.3 muestra el proceso de establecimiento de conexión MPTCP. Primero, la aplicación envía un Connect(), después de eso, la capa de transporte establece conexión con el destino correspondiente, quien está esperando a recibir las peticiones de conexión. Se aprecia que se lleva a cabo mediante el saludo a tres vías típico, como ya se mencionó anteriormente, excepto por la adición de la opción MPC, que implica el uso de MPTCP.

Una vez establecida la conexión, ambos extremos deben comunicar al otro las interfaces que tienen disponibles para así establecer los posibles sub-flujos que se usaran.

#### 2.1.1.2 - Iniciación del Sub-flujo


*Figura 2.4 – Iniciación de sub-flujo MPTCP*

La figura 2.4 muestra la iniciación de un nuevo sub-flujo, en ella se puede apreciar la presencia de JOIN en el segmento SYN. Para aumentar las posibilidades de que cada nuevo sub-flujo ocupe un *path* que no esté vinculado con los *paths* ya usados por los sub-flujos establecidos, cada dirección IP solo se emplea una vez por un único sub-flujo.

### 2.1.2 - Control de congestión

MPTCP redefine varios de los mecanismos que emplea TCP. Aunque dichos mecanismos se verán más detenidamente en la siguiente sección, aquí se resumirán brevemente.

El control de congestión permite al transmisor regular el *throughput* como respuesta a la disponibilidad de recursos de la red. En MPTCP, el control de congestión se lleva a cabo a nivel de sub-flujo, es decir, cada sub-flujo tiene su propia ventana de congestión independiente. Sin embargo, las diferentes ventanas de los sub-flujos de una conexión TCP no tienen por qué ser siempre independientes, pueden estar relacionadas de alguna manera para conseguir mejorar el rendimiento. Hay que añadir, que, en la sub-capa superior, el receptor MPTCP tiene una única ventana global compartida entre todos los sub-flujos establecidos.

Se han propuesto varios algoritmos [9] los cuales enlazan de diversas maneras las diferentes ventanas:

**Uncoupled:** la ventana de congestión de cada sub-flujo se comporta como una único conexión TCP.

**Fully Coupled**: cuando uno de los sub-flujos presente una congestión notable, el algoritmo intentara desviar la carga hacia otro *path* menos congestionado.

**Linked Increases:** con el anterior algoritmo, se utiliza uno u otro *path* indistintamente, y rara vez ambos. Este fenómeno se denomina *"flappiness"*. Este algoritmo se ha pensado para reducir el fenómeno *flappiness.*

**RTT Compensator:** está pensado para un caso general en el que el rtt (*roundtrip time*) no es igual para todos los *paths*.

### 2.1.3 - Reordenado de paquetes

En una conexión TCP existe la posibilidad de, que, debido a problemas en la red, cuando el retraso punto a punto puede variar mucho, los paquetes lleguen al receptor desordenados. Este puede ser el caso de las redes inalámbricas, en donde el dispositivo móvil puede cambiar el punto de acceso a Internet. El reordenado de paquetes hace que el receptor responda duplicando los ACK y esto puede provocar una pérdida de paquetes en el transmisor.

Como solución a este problema, y para distinguir claramente entre a perdida de paquetes debido a la congestión de la red y los procedentes del reordenado, TCP propone varios mecanismos. Varios de estos mecanismos han sido especificados por IETF en [9], [10], [11], [12]. Otros los definen los propios sistemas operativos.

En el caso de una conexión *multipath*, los paquetes que llegan desordenados pueden presentar diferentes características. La llegada de paquetes desordenados puede generar un problema para MPTCP, a la hora de reordenar los paquetes a nivel de conexión y no a nivel de sub-flujo, ya que los sub-flujos son independientes entre sí. MPTCP considera implementar algunas mecánicas similares a las que presentan TCP o las redes inalámbricas.

Algunos algoritmos como el algoritmo Eifel [12] y el DSACK [10], necesitan guardar información sobre el estado de la conexión. Esto es, la ventana de congestión (*cwnd*), el umbral del *Slow-Start* (*ssthresh*)… antes de retransmitir, y cuando una retransmisión no deseada se detecta, se restaura el estado de la conexión. Otros algoritmos, como el algoritmo Blanton-Allman [11] o el RR-TCP (Reordering-Robust TCP) [11] ajustan el número de ACK recibidos duplicados y después el transmisor considera cada paquete como perdido y lo retransmite.

En las siguientes sub-secciones se introducen algunos algoritmos estandarizados usados por TCP para reordenar paquetes.

#### 2.1.3.1 - Algoritmo Eifel

La figura 2.5 muestra cómo funciona el algoritmo [12]. El transmisor añade una opción *TCP timestamp* (tiempo del momento exacto en el que se envía) en cada segmento que envía y el receptor también inserta un *timestamp* en la confirmación correspondiente.

En caso de pérdida, el transmisor guarda el valor de la ventana de congestión (*cwnd*) y del umbral del *slow-start* (*ssthresh*). Posteriormente, se retransmite el segmento perdido, guardando el valor del *timesatmp* actual. Cuando el trasmisor recibe la confirmación del segmento retransmitido, se compara el timestamp guardado con el que esta insertado en el ACK. Si el primero es mayor, se considera que la retransmisión ha sido indeseada, y se restaura el valor del *cwnd* y *ssthresh*.

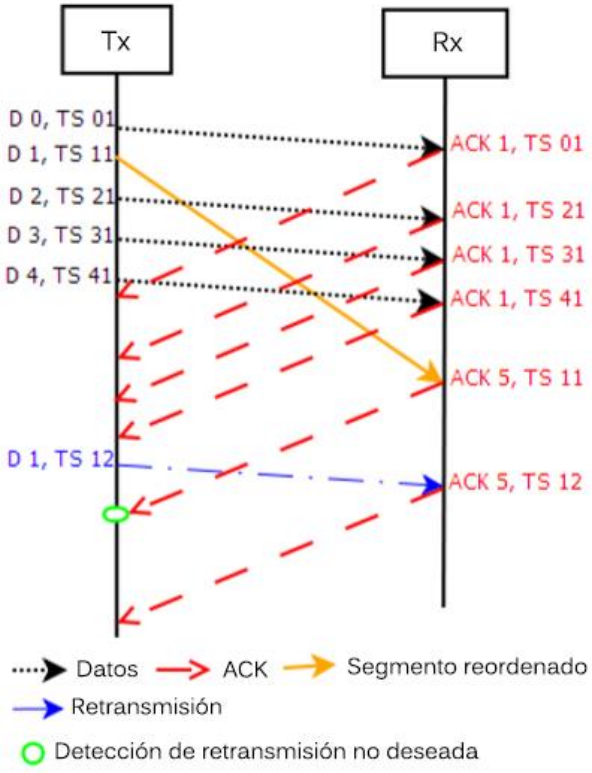


*Figura 2.5 – Algoritmo Eiffel*

### 2.1.3.2 - Algoritmo DSACK

La figura 2.6 muestra cómo funciona el algoritmo [10]. Este algoritmo está basado en la opción SACK (*Selective* ACK). En una recepción de un paquete que crea un espacio entre los números de secuencia, el receptor envía un ACK duplicado que contiene la opción SACK. EL primer bloque en la opción SACK hace referencia al segmento que ha disparado la confirmación duplicada. Tras tres ACK duplicados, el transmisor retransmite el segmento, guarda el valor de la ventana de congestión (*cwnd*) y entra en la fase *Congestion Avoidance*.

Posteriormente, cuando el transmisor detecta que el segmento retransmitido ha sido confirmado dos veces, asume que ha sido causa de una retransmisión no deseada, y comienza el mecanismo *DSACK Slow-Start* con el valor de *cwnd* que había guardado anteriormente.

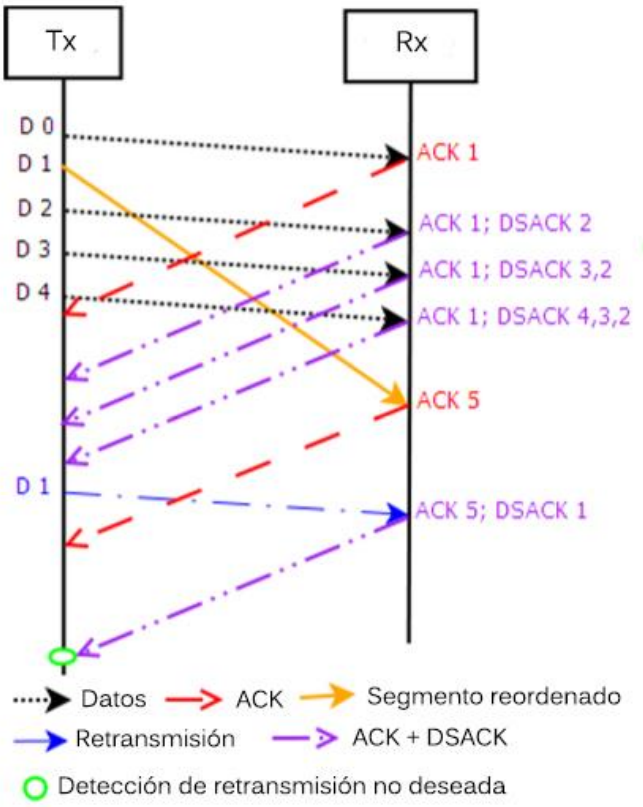


*Figura 2.6 – Algoritmo DSACK*

## 2.2 – Control de congestión

El control de congestión en un entorno *multipath* debe diseñarse para lograr una gestión óptima de los recursos disponibles. Como se ha mencionado anteriormente existen varios algoritmos diseñados para lograr distribuir el tráfico entre los diferentes *paths* disponibles. Estos algoritmos aumentan el valor de la ventana de congestión (*cwnd*) como respuesta a las confirmaciones de cada paquete en ese *path* y disminuye su valor para reaccionar a las perdidas y posibles problemas que se presenten en ese camino. La cantidad que aumenta o disminuye la ventana depende de valores como la ventana de congestión global o el *roundtrip time* de todos los otros *paths* en uso. Escogiendo adecuadamente estos aumentos o disminuciones de la ventana de congestión el algoritmo lograra una mayor eficacia. Hay que añadir que el algoritmo de control de congestión debe tener en cuenta las metas que debe cumplir MPTCP y que ya se vieron anteriormente.

Para los siguientes algoritmos se considerará que se dispone de múltiples *paths* $r = 1 \dots N$ donde cada uno dispone de su propia ventana de congestión. La ventana $w_r$ será la correspondiente al flujo $r$ y $\sum_{i=1}^{N} w_i = w$ será el tamaño de la ventana global.

**ALGORITMO FULLY COUPLED**

Se incrementa $w_r$ en $1/w$ por cada confirmación en el *path* $r$.

Se disminuye $w_r$ en $\max(w_r - w/b, 1)$ por perdida en el *path* $r$. Se puede usar $b = 2$ para imitar a TCP.

Cuando un *path* presenta un problema de congestión, este algoritmo tratará de desviar la carga a otro flujo para que de esta manera se reduzca la congestión. El problema de este algoritmo se presenta en las perdidas prolongadas en un mismo flujo, lo que implicara una reducción de dicha ventana, manteniéndose en ese estado hasta que se vuelva a una situación estable.

**ALGORITMO LINKED INCREASES**

Se incrementa $w_r$ en $a/w$ por cada ACK en el *path* $r$.

Se disminuye $w_r$ en $\mathrm{w_r}/2$ por perdida en el *path* $r$.

Donde $a$ se elige para controlar la agresividad del flujo.

Este algoritmo presenta una reducción del efecto *flappiness* (visto anteriormente) que presenta el algoritmo anterior. A cambio, se sufre un ligero decremento en la gestión de los recursos, evitando el reparto poco equitativo entre diferentes sub-flujos.

**ALGORITMO RTT COMPENSATOR**

Se incrementa $w_r$ en $\min(a/w,\ 1/w_r)$ por cada confirmación en el *path* $r$.

Se disminuye $w_r$ en $w_r/2$ por cada perdida en el *path* $r$.

Como vimos en la sub-sección anterior, este algoritmo está diseñado para situaciones en las que el valor del *rtt* (*roundtrip time*) pueda variar mucho. El algoritmo es una ligera variación del anterior.

**ALGORITMO UNCOUPLED**

Se incrementa $w_r$ en $1/w_r$ por cada confirmación en el *path* $r$.

Se disminuye $w_r$ en $w_r/2$ por cada perdida en el *path* $r$.

En este último caso, cada flujo es independiente del resto y se comporta como una única conexión TCP.

Para los casos 2 y 3 se ve la expresión *a*, que representa el factor de agresividad. Asignar un valor a este factor incumpliría las metas que debe cumplir MPTCP (y que vimos en el apartado), por tanto, se calcula y se le asigna un valor en cada instante mediante la siguiente expresión. Sin embargo, no cumple del todo con la condición 3, ya que no permite reducir la carga en los sub-flujos más congestionados de la conexión.

$$a = w \left( \frac{\max_r \frac{\sqrt{w_r}}{RTT_r}}{\sum_r \frac{w_r}{RTT_r}} \right)^2$$

## 2.3 - Scheduler

A la hora de enviar datos, al tener varios caminos disponibles el transmisor debe elegir por cuales transmitir en cada momento. Pero una mala elección puede hacer que el rendimiento que se obtiene empleando múltiples caminos y MPTCP sea peor que emplear un único camino con TCP. Es por eso que se tiene la necesidad de implementar un *scheduler* apropiado.

Cuando el transmisor MPTCP quiere enviar datos, el transmisor debe tomar varias decisiones: Primero, cual o cuales sub-flujos están disponibles y podrían ser usados para enviar datos. Esta elección se realiza gracias al control de congestión al mantener cada ventana de congestión independiente para cada sub-flujo. Segundo, si existen varios sub-flujos para enviar con una ventana que lo permita, el *scheduler* debe elegir por cual enviara. Y tercero, tras seleccionar el sub-flujo por el que se enviara, el *scheduler* determinara la cantidad de datos para enviar.

La implementación por defecto en Linux es la siguiente:

Cuando existen más de un sub-flujo disponible, los datos se transmiten por aquel que tenga un menor *srtt* (*smoothed roundtrip time*).

Esta implementación parece lógica y buena. El *srtt* hace referencia al tiempo desde que se envía el paquete y llega la confirmación del mismo, por lo tanto, se está seleccionado el sub-flujo más rápido para enviar los paquetes de datos por él.

Pero pueden surgir problemas. Una de los objetivos del control de congestión es evitar los caminos congestionados, sin embargo, si son más de uno los sub-flujos disponibles para enviar, con una ventana de congestión (*cwnd*) apropiada, esta meta no se logrará sin un buen *scheduler*. Se puede dar el caso en el que un *scheduler* basado solo en el *srtt* resulta inconsistente. En [14] se presenta un escenario en el que ocurre esto y ante una situación en la que ambos sub-flujos tienen una ventana de congestión disponible para él envió, el *scheduler* elige el camino congestionado.

En [14] se nos presenta la idea de emplear un *scheduler* que se base en la ocupación de cada sub-flujo en cada momento. Pero estimar la disponibilidad de cada camino de una red es difícil, ya que las capacidades de cada camino están constantemente cambiando. Sin embargo, este problema no es nuevo, en TCP se intenta estimar la capacidad dinámicamente para el control

de congestión. Es por eso que se pueden emplear técnicas de TCP para estimar la capacidad disponible de cada sub-flujo.

Los parámetros de los que se disponen para hacer la estimación son la ventana de cada sub-flujo (*cwnd*), el *ssthresh* (*slow start threshold*) y todos los valores relacionados con el RTT (*roundtrip time*) como *srtt*, RTO…

Para estimar la capacidad de cada camino el método a emplear debe ser estable. En [14] se explica con ejemplos porque no sirve un método que solo emplee el RTT o el valor del *cwnd*. El método a emplear debe basarse en varios de los parámetros disponibles.

**if** (a loss has been detected since last run so that $ss\_threshold$ has been updated) **then**
  $Max = 2 * ss\_threshold$
  $Min = ss\_threshold$
  $Estimated\_path\_capacity = \frac{1}{2} * (Max + Min)$
**else if** $(Num\_of\_outstanding \geq Estimated\_path\_capacity)$ **and** $(Num\_of\_outstanding < Max)$ **then**
  $Min = Num\_of\_outstanding$
  $Estimated\_path\_capacity = \frac{1}{2} * (Max + Min)$
**else if** $(Num\_of\_outstanding \geq Max)$ **then**
  $Max = Cwnd$
  $Min = Num\_of\_outstanding$
  $Estimated\_path\_capacity = \frac{1}{2} * (Max + Min)$
**end if**

*Figura 2.7 – Algoritmo para estimar la capacidad disponible en cada sub-flujo*

Una vez que se tiene la estimación de la ocupación de cada camino, se puede pasar al *scheduler*.

Cuando un paquete esté listo para enviarse, el transmisor determina que sub-flujos están disponibles. El valor de $Occupied_i$ se calcula para cada flujo $i$. Si el paquete a enviar es realmente una retransmisión, no se enviará por el mismo sub-lujo que el original. En cualquier otro caso, el sub-flujo con $Occupied_i < \gamma$ se podrá emplear para enviar los datos. En caso de que sean varios los que cumplen eta condición y no estén congestionados, se usara aquel con un menor *srtt*. Por último, en caso de que todos los sub-flujos disponibles estén congestionados, se elegirá el que tenga menor valor de $Occupied_i$ .

Each time the scheduler runs:
$Min\_srtt = 0xFFFFFFFF$
$Min\_occupied = 0xFFFFFFFF$
$Num\_uncongested\_path = 0$
$Num\_available\_path = 0$

**for** each subflow $i$ **do**
  **if** (next packet is a retransmission) **and** (it has been transmitted on subflow $i$) **then**
    continue
  **end if**
  **if** $(Num\_of\_outstanding_i \geq Cwnd_i)$ **then**
    continue
  **end if**
  $Occupied_i = (Num\_of\_outstanding_i + 1)/Estimated\_path\_capacity_i$
  **if** $(Occupied_i > \delta)$ **then**
    continue
  **end if**
  $Num\_available\_path$++
  **if** $(Occupied_i \leq \gamma)$ **then**
    $Num\_uncongested\_path$++
  **end if**
**end for**

**if** $(Num\_uncongested\_path > 0)$ **then**
  **for** each uncongested subflow $i$ **do**
    **if** $(Srtt_i < Min\_srtt)$ **then**
      $Min\_srtt = Srtt_i$
      $Selected\_subflow = i$
    **end if**
  **end for**
**else if** $(Num\_available\_path > 0)$ **then**
  **for** each available subflow $i$ **do**
    **if** $(Occupied_i < Min\_occupied)$ **then**
      $Min\_occupied = Occupied_i$
      $Selected\_subflow = i$
    **end if**
  **end for**
**else**
  $Selected\_subflow = NULL$
**end if**

*Figura 2.8 – Algoritmo del scheduler*

## 2.4 - LTE

LTE (*Long Term Evolution*) ha sido diseñado para soportar únicamente servicios de conmutación de paquetes. Su objetivo es proporcionar una conexión entre los usuarios (UE, *user equipment*)

y la red de datos (PDN, *packet data network*) sin ningún problema durante la movilidad del usuario.

LTE engloba la evolución del acceso radio de UMTS (*Universal Mobile Telecomunications System*) hacia el E-UTRAN (*Evolved UTRAN*), acompañado de otros aspectos englobados en el término *System Architecture Evolution* (SAE), en los cuales se incluye la red *Evolved Packet Core* (EPC). Juntos, LTE y SAE constituyen el *Evolved Packet System* (EPS).

El EPS utiliza el concepto de portadoras EPS (*EPS bearers*) para enrutar trafico IP desde el Gateway situado en el PDN y el UE. Una portadora es un flujo de paquetes IP con una determinada *Quality of Service* (QoS) entre el Gateway y el UE. Son el E-UTRAN y el EPC los encargados de asignar o liberar portadoras según corresponda a las aplicaciones.

## 2.4.1 - Resumen de la arquitectura

El EPS provee al usuario de una conexión IP con el PDN para poder acceder a internet, además de ofrecer servicios como Voz sobre IP (*VoIP*). Una portadora EPS se suele asociar a una determinada QoS, pudiendo asignar múltiples portadoras a un usuario con el fin de ofrecerle varias QoS o conectividad con diferentes PDNs. La red también ofrece seguridad y privacidad frente al posible uso fraudulento de la misma.

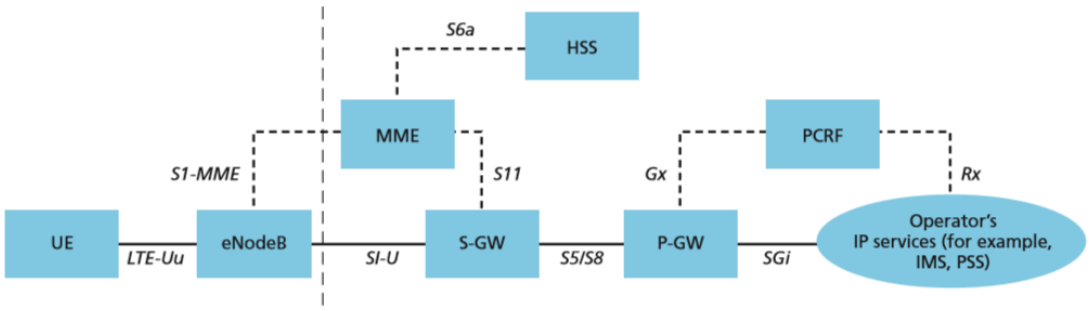


*Figura 2.9 - Arquitectura de red LTE*

Esto se consigue gracias al uso de diferentes elementos de la red EPS. En la figura 2.8 se puede ver un esquema de la arquitectura de red, donde se incluyen los elementos de red y las interfaces estandarizadas.

A más alto nivel, la red está constituida por el CN (EPC) y el acceso a la red E-UTRAN, en donde el CN está formado por varios nodos lógicos, pero el acceso a la red solo es llevado a cabo por uno, el *evolved NodeB* (eNodeB), el cual se conecta a los UEs. Las interfaces estandarizadas permiten la interoperabilidad entre equipos de diferentes vendedores, facilitando así la labor del operador de red.

La funcionalidad dividida entre el EPC y el E-UTRAN se muestra en la figura 2.10.

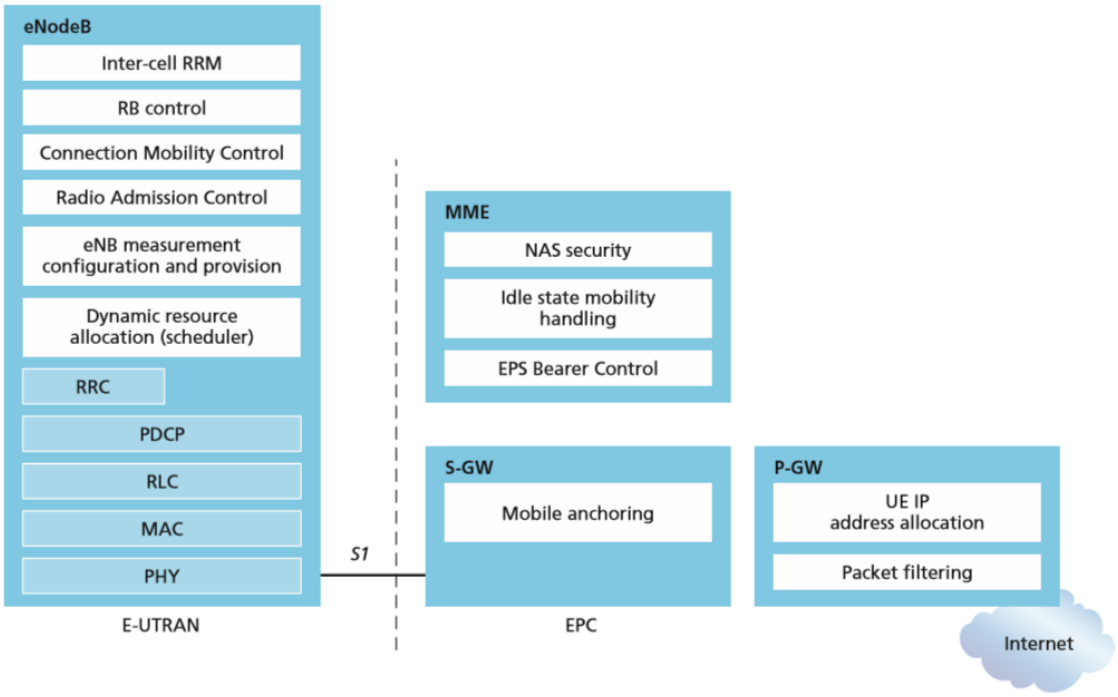


*Figura 2.10 –Funcionalidad entre el EPC y E-UTRAN*

## 2.4.2 – Núcleo de la red

El núcleo de la red (llamado EPC y SAE) es el responsable del control de los UE y del establecimiento de portadores. Los principales nodos son los siguientes:

- PDN Gateway (P-GW)
- Serving Gateway (S-GW)
- Mobility Management Entity (MME)

Junto a estos nodos, el EPC incluye otros nodos lógicos y funcionalidades como el Home Subscriber Server (HSS) y el Policy Control and Charging Rules Function (PCRF),

A continuación, se detallan brevemente estos nodos que ya se vieron en la figura 2.9. En [15] se pueden ver las funcionalidades con mayor grado de detalle.

**PCRF** – El Policy Control and Charging Rules Function es el encargado de las tomas de decisiones del control de normas además de encargarse de la autorización de QoS a un determinado flujo de datos.

**HSS** – El Home Subscriber Server contiene los datos de suscripción de los usuarios al SAE además de las restricciones de roaming y los perfiles de QoS del usuario. También contiene información acerca de a que PDN puede conectarse el usuario.

**P-GW** - El Gateway del PDN es el responsable de asignar las direcciones IP a los UE.

**S-GW** – Mediante el Serving Gateway todos los paquetes de datos de los usuarios son transferidos.

**MME** – El Mobile Management Entity es el nodo de control que procesa la señalización entre el UE y el CN. El protocolo que corre entre el UE y el CN se conoce como Non Access Stratum (NAS).

### 2.4.3 - El acceso a la red

Como se muestra en la figura 2.11, el acceso a la red se basa en una serie de ENodeBs. Estos nodos están interconectados entre ellos mediante una interfaz llamada X2 y al EPC mediante la interfaz S1. El protocolo que se usa entre los ENodeB y los UE se llaman AS Protocols. EN [15] se puede ver en detalle la funcionalidad de estos protocolos.

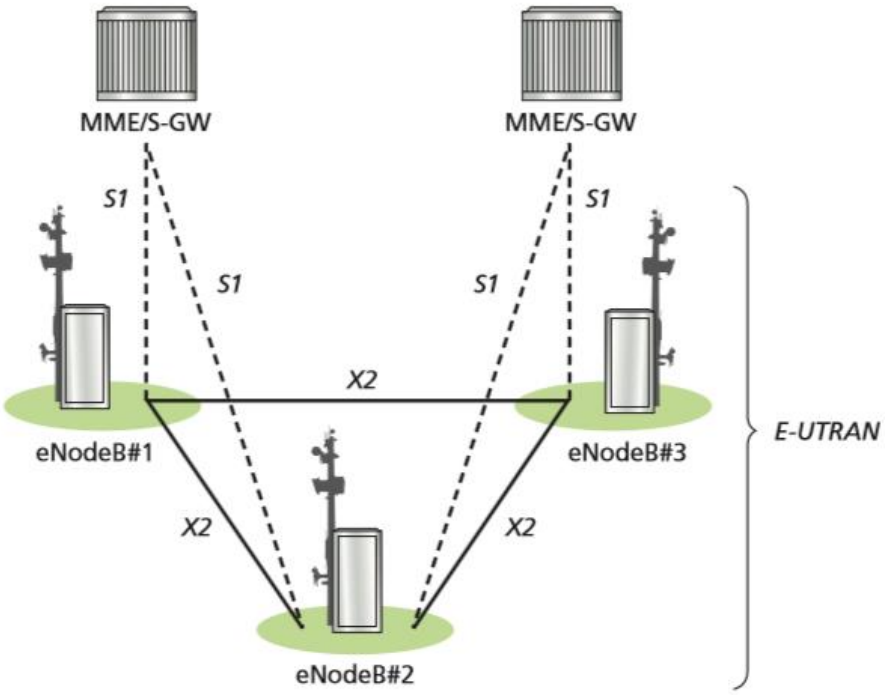


*Figura 2.11 –Acceso a la red*

## 2.4 - WIFI

Esta subsección tiene como objetivo dar al lector los conocimientos básicos sobre la tecnología Wifi o 802.11. No pretende ser una sección extensa que cubra todo el tema, más bien, dar a conocer los conceptos que se verán más adelante en los escenarios y topologías.

La especificación IEEE 802.11 (ISO/IEC 8802-11) es un estándar internacional que define las características que debe cumplir una *Wireless local area network* (WLAN o red de área local inalámbrica). Es decir, empleando wifi se pueden crear redes de área local inalámbricas de alta velocidad, siempre que los dispositivos se encuentren a una distancia razonable del punto de

acceso. En la actualidad estamos acostumbrados al uso de wifi en el día a día, portátiles, *smarthpones*, *tablets*, televisores… y una multitud de dispositivos permiten la conexión a una red wifi.

802.11 está compuesto por una gran cantidad de recomendaciones (las características letras que acompañan al nombre) que se van actualizando con el tiempo, otorgando nuevas prestaciones a las redes wifi.

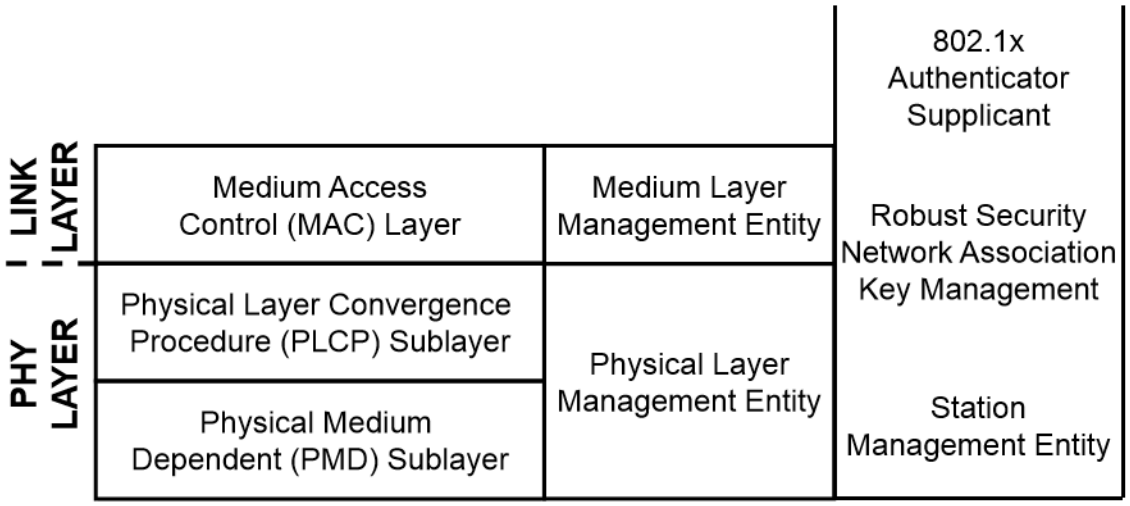


*Figura 2.12 – Modelo de referencia*

El modelo de referencia de la especificación define una capa física y una capa MAC, o de enlace, que serán las que se vean posteriormente. A continuación, se repasa brevemente una característica básica de la red como las diferentes topologías y configuraciones.

Existen tres topologías básicas más una adicional que ha incorporado 802.11ad:

- **IBSS** (Independet Basic Service Set): en la que las estaciones se comunican directamente sin necesidad de un punto de acceso.
- **BSS** (Basic Service Set): en donde existe un punto de acceso quien conecta a las estaciones a una red.
- **ESS** (Extended Service Set): una extensión del caso anterior, en el que varias BSSs conjuntas se conectan a una red. Cada BSS tiene su punto de acceso.
- **PBSS** (Personal BSS): se trata de una BSS a escala personal.

## 2.4.1 – Capa física (PHY)

La capa física de 802.11 presenta una arquitectura que se divide a su vez en dos sub-capas diferentes:

- **PMD** (Physical Medium Dependent): quien se encargar de gestionar las características intrínsecas al medio inalámbrico, así como de gestionar los métodos de transmisión (v.gr., modulación, codificación…)

- **PLCP** (Physycal Layer Convergence Procedure: de forma resumida, adapta las PDU de la capa MAC a SDU para la paca PMD, permitiendo así la definición de un protocolo MAC genérico.

Las diferentes recomendaciones han especificado diferentes posibilidades para la capa física, es decir, métodos de modulación, medios de transmisión, bandas frecuencias que emplea…

### 2.4.2 – Capa MAC

En esta capa surge la necesidad de arbitrar el acceso a un medio compartico, como es el que se trata. Para ello se presenta un acceso llamado DCF (Distributed Coordination Function) basado en el mecanismo CSMA (Carrier Sense Multiple Access) con la opción CA (Collision Avoidance).

El funcionamiento básico del sistema de acceso es el siguiente: A la hora de transmitir, un nodo escucha primero el medio informándose de esta manera de si alguien está transmitiendo. En caso de que el canal este libre durante un periodo de tiempo DIFS, se transmite. En caso contrario, si el medio está ocupado, se espera un tiempo DIFS y se vuelve a comprobar que el medio esta libre, se espera un tiempo aleatorio y se procede a la transmisión.

En el caso de la recepción de datos, funciona de manera similar. Cuando se recibe un paquete de datos, se espera un tempo SIFS tras el cual se envía la confirmación correspondiente. En caso de no recibir ACK en el tiempo establecido para ello, el nodo da la trama por perdida, y se retransmite.

En esta capa se pueden dar situaciones como la denominada “Terminal Oculto”, o la necesidad de transmitir datos en tiempo real. Situaciones que no se solucionan con el mecanismo básico explicado, pero que si tienen solución implementada que no se verá en el capítulo.

# 3

# Desarrollo en ns-3

Este capítulo presenta el simulador ns-3, con el cual se han desarrollado las simulaciones. Se tratarán brevemente sus características principales, así como una visión algo más detallada de los diferentes módulos y componentes que se usan en las simulaciones del siguiente capítulo, junto con los cambios realizados a algunos módulos para adaptarlos a los intereses deseados.

## 3.1 - Introducción a NS3

Fue entre los años 1995-1997 cuando este simulador (o más bien, una versión antigua del mismo) vio la luz. Se conocía como ns-1, y estaba escrito en C++ y usaba Tcl (Tool Command Language) para describir los escenarios. Un año después, se unieron al proyecto varias entidades que apoyaron y contribuyeron al desarrollo de una segunda versión, que se conocería como ns-2. Se mantuvo la parte de C++, pero el Tcl se sustituyó por OTcl (Obejct Tcl), una versión de Tcl orientada a objetos.

El proyecto de ns-3 comenzó en 2006 como un proyecto de software libre, y en 2008 se lanzó la versión 3.1. Actualmente el simulador se encuentra en la versión 3.25, publicada en marzo del 2016.

El simulador ns-3 (Network Simualtor 3) es un simulador de red de eventos discretos diseñado para el uso educativo y para la investigación.

Algunas de las características que ns-3 puede ofrecernos son:

- Ns-3 es un software libre y el proyecto pretende seguir manteniendo este carácter de entorno abierto para que todos los investigadores puedan contribuir compartiendo su software

- Ns-3 no presenta compatibilidad hacia atrás, en otras palabras, no se trata de una extensión de ns-2, es un simulador nuevo y diferente. Aunque ambos estén escritos en C++, ns-3 no soporta los APIs de ns-2. Aunque muchos de los modelos del simulador antiguo ya han sido adaptados a este nuevo simulador.

Hay que añadir que el proceso de aprendizaje de este simulador puede resultar, en ocasiones, difícil, pues se debe introducir y explicar al usuario varios conceptos y prestaciones de las que ns3 dispone.

Como ya se ha dicho, ns-3 se ha desarrollado para proveer de una plataforma abierta para propósitos de investigación. Se podría decir, de forma breve, que los modelos de ns-3 simulan como las redes de paquetes de datos trabajan y que rendimiento tienen, dando al usuario la capacidad de simular experimentos. Muchos de estos experimentos no pueden llevarse a cabo en sistemas reales o simplemente resultan más fáciles de simular, es por eso que se puede estudiar el rendimiento de la red, y ver cómo trabaja bajo determinadas situaciones altamente controladas.

Es cierto que existen varias alternativas a la hora de hacer simulaciones de red. A continuación, trataremos algunas de las características que presenta ns-3:

- Ns-3 presenta una variedad de librerías que pueden combinarse juntas o con otras externas. Mientras otras plataformas ofrecen una única e integrada interfaz de usuario gráfica, ns-3 tiene un comportamiento modular.
- A pesar de que ns-3 se use principalmente en Linux, tiene soporte Windows y otros sistemas.
- Ns-3 no está apoyado por ninguna empresa o compañía.

## 3.2 - Características

ns-3 es un simulador de redes basado en eventos discretos cuyo núcleo y diversos modelos están implementados en C++. Ns-3 es una librería la cual puede estar vinculada dinámica o estáticamente a un programa principal en C+ que defina la topología de la simulación, características de la misma y comience la simulación. Además, ns-3 permite exportar prácticamente todas sus APIs a Python, permitiendo a los programas en Python a importar un módulo ns3.

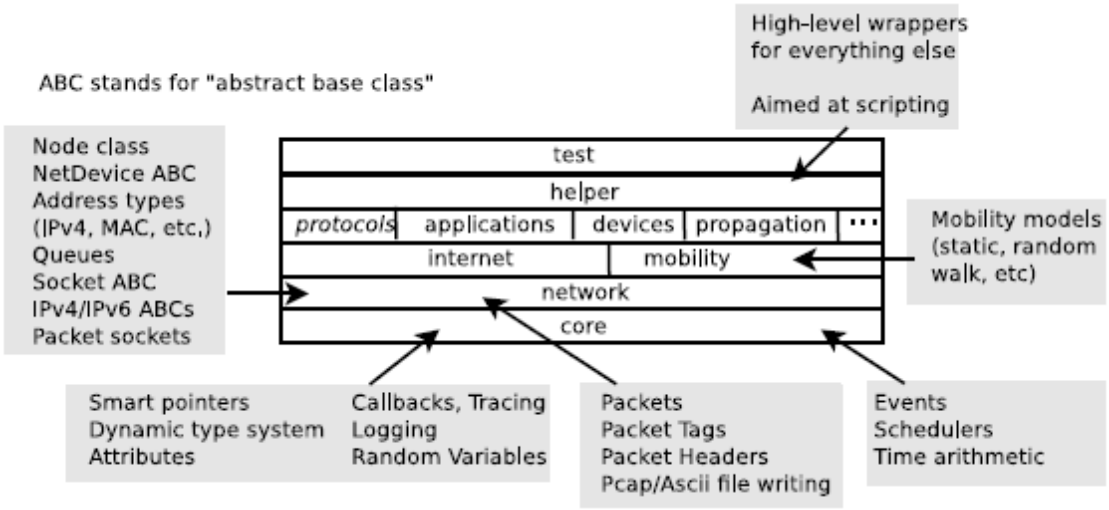


*Figura 3.1 – Organización software de ns-3*

El código fuente de ns-3 esta, en su mayoría, organizado en el directorio src. Como se aprecia en la figura 3.1, los módulos solo tienen dependencias de otros módulos que estén sobre ellos.

EL núcleo del simulador (*core*) lo forman los componentes que son comunes en todos los protocolos, hardware y modelos de entorno. El núcleo esta implementado en *src/core*. Los paquetes son objetos fundamentales en un simulador de red, como cabe esperar, y están implementados en *src/network*. Estos dos módulos por si mismos pretenden abarcar el núcleo de un simulador genérico que pueda ser empleado en diversas y variados tipos de redes, no solo en aquellas basadas en internet.

Los módulos que se ven por encima de estos dos en la figura 3.1 son independientes de los modelos de red o de los dispositivos de red, y cubren los diferentes módulos del simulador. Junto al núcleo de ns-3 ya descrito, existen otros dos módulos que complementan a la API del núcleo. Todos los programas ns3 pueden acceder a las APIs directamente, o si se prefieren hacer uso de un mecanismo llamado *helper* API, que aporta los enlaces convenientes a las llamadas a las API a bajo nivel.

En las siguientes sub-secciones se verán los módulos básicos que se han empleado en la simulación, así como las adaptaciones que se han aplicado en los mismos.

## 3.3 – Componentes básicos

Aunque ns-3 se compone de gran variedad de módulos [17], y cada uno de ellos con multitud de ficheros C++ donde se describe su funcionalidad, en este apartado veremos con más detalle solo aquellos que se emplean más en el desarrollo del trabajo, y por tanto son considerados básicos. Estos son los módulos que abarcan TCP/IP, Wifi, movilidad y los elementos de red básicos de ns-3.

### 3.3.1 – Network module

Este módulo se compone de los elementos básicos de red, como vimos anteriormente, para que puedan proveer de funcionalidades básicas a cualquier topología y simulación. Este módulo contiene los elementos*: Packets, Error Model, Node, NetDevices…* Todos estos elementos están detallados en [17], esta sección se centrará en los elementos de los nodos, y dispositivos de red.

**Clase Node**: se trata de la clase fundamental para las topologías, en donde se instancia una lista de *ns3::Application*, una lista de *ns3::NetDevice* y una lista de *ns3::ProtocolHandler*, como se ve en la figura 3.2. Cabe destacar que inicialmente estas listas están vacías.

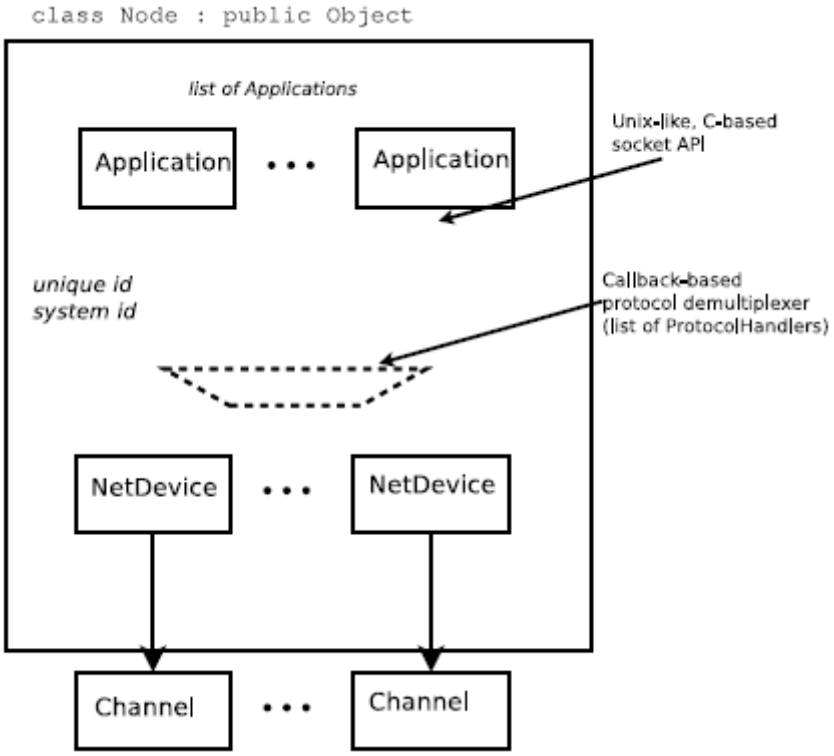


*Figura 3.2 - Arquitectura de nodo a alto nivel*

### 3.3.2 – PointToPoint Model

El modelo *PointToPoint* es simplemente un enlace de datos entre dos dispositivos *PointToPointNetDevice* sobre un *PointToPointChannel*.

El *PointToPointNetDevice* presenta los siguientes atributos:

- *Address*: La dirección del dispositivo (*ns3::Mac48Address*) deseado.
- *DataRate*: La tasa de datos (*ns3::DataRate*) del dispositivo.
- TxQueue: La cola de transmisión (*ns3::Queue*) que emplee el dispositivo.
- InterframeGap: El tiempo de espera (*ns3::Time*), opcional, entre *"frames"*.
- Rx: La fuente para "rastrear" los paquetes recibidos. Rastrear o *"trace" es la forma de extraer los datos de la simulación.*
- *Drop*: La fuente para "rastrear" los paquetes descartados.

El *PointToPointChannel* es la vía por la que se conectan y transmiten dos dispositivos.

### 3.3.3 – Mobility Model

Los objetos MobilityModel en ns3 rastrean la evolución de la posición respecto a un sistema de coordenadas cartesiano. El modelo se suele incluir en el *ns3::Node*. De esta forma se puede definir perfectamente a posición de cada nodo en cada instante, sobre todo para definir dispositivos móviles o dispositivos wifi.

### 3.3.4 – Internet Models

El modulo define los elementos de internet tales como la pila de internet (*InternetStack*), los protocolos IP, TCP y direccionamiento. En esta sección se verá brevemente los protocolos de IP y TCP, aunque este segundo será sustituido por MPTCP, que se verá en la siguiente sección.

#### 3.3.4.1 – InternetStack

Los nodos de internet no son una sub-clase del nodo, como uno podría pensar, simplemente son nodos (Node) al que se le agrega la información de la pila de protocolos. Esto puede hacerse manualmente o mediante la ayuda del helper *InternetStackHelper::Install*. En [17] se detalla el funcionamiento de esta función.

#### 3.3.4.2 – TCP

La implementación de TCP de ns3 presenta dos clases básicas:

**Clase TcpSocket**: está definida en *src/internet/model/tcp-socket. {cc,h}.* Esta clase implementa los diferentes atributos del Socket, que podrán reutilizarse en diferentes implementaciones. Por ejemplo, el atributo *InitialCwnd*.

**Clase TcpSocketFactory**: Se usa por las aplicaciones para crear sockets TCP.

Además de estas dos clases base que presenta el modulo, también existen otras clases que definen el funcionamiento, pero la explicación de las mismas las obviaremos ya que en este trabajo se ha sustituido TCP por el módulo MPTCP que se verá con detenimiento en la sección posterior.

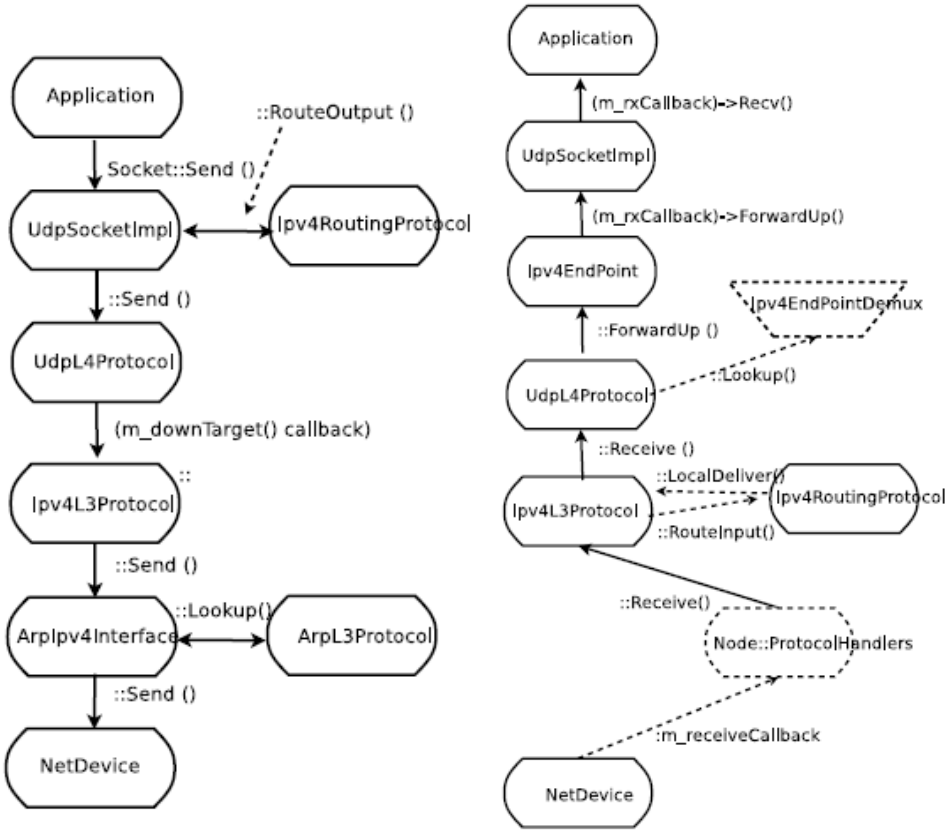


*Figura 3.3 – Envió y recepción de un paquete TCP*

La figura 3.3 muestra el flujo de un paquete sobre unos nodos que estén usando arquitectura TCP.

## 3.3.5 – Wifi Model

Al igual que en la realidad los ordenadores poseen diferentes interfaces, los nodos de ns-3 pueden contener una variedad de diferentes objetos *NetDevice*, y en ambos casos uno de esos objetos/interfaces puede ser la wifi. En esta sección se describe el objeto *WifiNetDevice* y como crear redes 802.11.

El modelo wifi para ns-3 incluye diversas características y prestaciones, tales como: diferentes modelos de pérdidas de propagación (Nakagami, Rayleight…), diferentes implementaciones de la capa física…todas ellas se pueden consultar en [17], esta sección se centrará en las clases fundamentales del módulo, así como en una vista general del funcionamiento.

Al examinar el código de un escenario, se puede apreciar el uso de las siguientes clases:

**YansWifiChannelHelper**: a pesar de su inusual nombre, se puede ver con bastante frecuencia. Este *Helper* es el encargado de crear un canal wifi con un modelo de pérdidas de propagación y retraso de propagación por defecto.

**YansWifiPhyHelper**: de forma similar al anterior, esta clase crea un objeto el cual creara instancias de la capa física, dando la opción de incluir un modelo de error (*ErrorRateModel*) suplementario o un modelo de movilidad (*MobilityModel*).

**NqosWifiMacHelper** y **QosWifiMacHelpe**r: ambas clases sirven para crear instancias de un objeto *ns2::WifiMac*, los cuales son configurados con parámetros típicos como la clase de MAC.

**WifiHelper**: será la clase encarga de crear los WifiNetDevices que se asociaran a los nodos. Para crearlos será necesario haber creado tanto la capa física como la MAC.

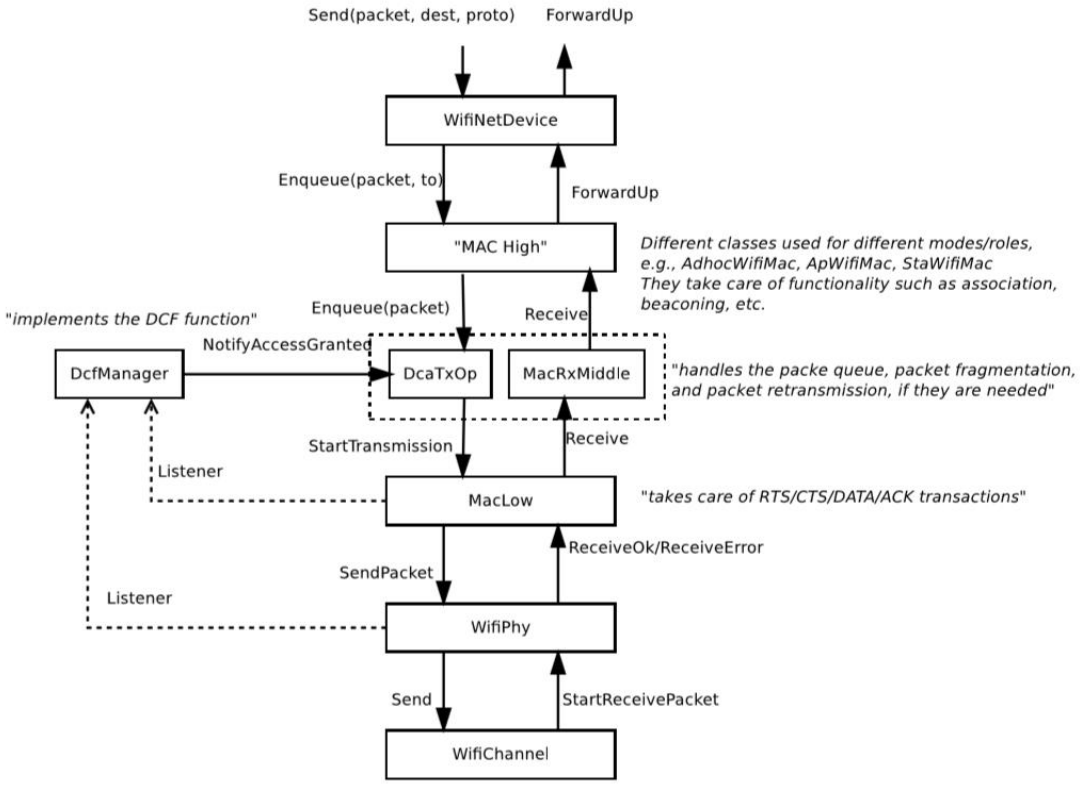


*Figura 3.4 – Arquitectura de un WifiNetDevice*

## 3.3.6 – Modulo LTE

En el capítulo anterior se veía una breve introducción a LTE, explicando, *grosso modo*, como funcionaba. Explicar cómo funciona el modelo LTE para ns-3 sería demasiado complejo y extenso. Por ello se verá muy brevemente donde residen las funcionalidades que ya vimos, y de forma esquemática su funcionalidad. [22] ofrece una lectura en mayor detalle de todos los aspectos de la implementación del modelo.

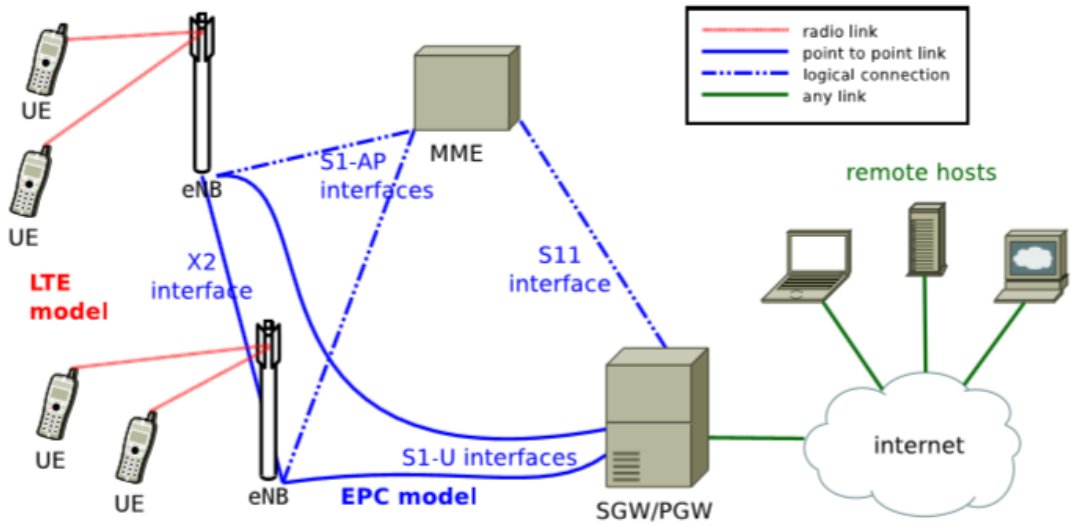


*Figura 3.5 – Resumen del modelo LTE-EPC*

El modelo consta de dos componentes principales:

- El modelo LTE, que incluye el LTE Radio Protocol. Esta entidad recae totalmente sobre el UE y los nodos eNB.

- El modelo EPC, que incluye las interfaces de la red, protocolos y entidades. Estas entidades residen en los nodos SGW, PGW y MME.

El modelo LTE se divide en el UE y el eNB, los cuales, a su vez, se pueden dividir de la siguiente forma:

- Arquitectura del UE: por un lado, presenta el modelo de la arquitectura del LTE radio protocolo, la cual contiene un plano de datos y un plano de control. Además, existe un modelo para la arquitectura de la capa física y el canal.
- Arquitectura de eNB: Al igual que el anterior, existe por un lado la arquitectura del *LTE radio protocol* del eNB (dividido de nuevo en plano de datos y de control). Y por otro lado el modelo de la capa física y el canal del eNB.

El modelo EPC, como ya se ha comentado, contiene los protocolos del SGW, PGW y MME. En esta ocasión no se verán diferentes arquitecturas para cada nodo, si no que veremos el EPC en conjunto. En donde están definidos el plano de control y el plano de datos. Dentro de cada una de estas arquitecturas se encuentra todas las interfaces modeladas del EPC.

### 3.3.7 – Modelo de Error

Aunque el *ErrorModel* está dentro del módulo de red, se tratara en esta sección con un poco más de detalle. Esta clase (o varias clases como se verá más adelante) pretenden simular el error en un enlace mediante diversos mecanismos.

En principio se presentan varios tipos de modelo de error, todos ellos asociados, normalmente, al *NetDevice*. Las diferentes clases de error son:

- RateErrorModel
- ListErrorModel
- ReceiveListErrorMode
- BurstErrorModel

En el trabajo se ha empleado la clase *RateErrorModel*, así como otra que fue creada partiendo de esta. El uso de esta clase es sencillo, pues como parámetro solo es necesario la tasa de error deseada a aplicar en el enlace e instalar la propiedad en el nodo correspondiente.

Hay que destacar la posibilidad de resetear, activar, desactivar o comprobar si está activado el modelo de error que nos ofrece la clase *ErrorModel*.

Por último, aunque existen clases (*NetDevice*, *CsmaNetDevice*…) que ya implementan este modelo de error como atributo, el *ErrorModel* (o cualquiera de los citados) puede usarse en cualquier lugar donde se transmitan datos.

#### 3.3.7.1 – MatrixErrorModel

Esta clase fue creada con el fin de aplicar múltiples y diferentes tasas de error a cada enlace del escenario, simplificando el código y pudiendo cambiar el escenario para las diferentes simulaciones. Esta clase se basas en *RateErrorModel*, pero crea una matriz en la que en la primera columna se introduce el nodo a aplicar el error y en la segunda la tasa de error correspondiente.

Para este trabajo, la clase fue modificada, añadiendo interfaces, es decir, ahora se tendrá el nodo en la primera columna, la interfaz correspondiente en la segunda y la tasa de error en la última.

El funcionamiento de la clase es sencillo. A la hora de aplicar el error en el nodo en el que nos encontremos, la clase busca en la matriz, comparando la ID del nodo actual con las diferentes identificaciones de los nodos almacenados, repitiendo el mismo proceso con la interfaz. Por último, una vez encontrado la interfaz y nodo correctos, se aplica la tasa de error que le corresponde.

## 3.4 – Modulo MPTCP

A pesar de ser un componente más del simulador ns-3, es el más importante en este trabajo, es por ello que se verá con mayor detalle que los anteriores. En un principio, se trabajó sobre una versión desarrollada en 2012 del protocolo MPTCP sobre la versión 3.13 del simulador, adaptándola para la versión 3.17. La implantación creaba una nueva clase llamada MPTCP que convivía junto a TCP, en otras palabras, era un módulo nuevo e independiente del módulo NETWORK (que ya se ha visto). Pero esta versión dejo cabos sueltos y funcionalidades

incompletas, es por ello que se cambió a una versión más reciente, implementada para la versión 3.21 del simulador, en donde se trata MPTCP como parte de TCP, es decir, dentro del mismo modulo NETWORK. Esta nueva versión, más completa que la anterior, presento varios detalles que hubo que adaptar (como el *Schedule* o el control de congestión) para el desarrollo de este trabajo, y que se verá en capítulos posteriores. Al ser una nueva implementación, para el desarrollo del trabajo hubo que estudiarla de nuevo, olvidando la anterior y comprender su funcionamiento, esto se explicará a continuación.

En este trabajo lo que nos interesa saber del módulo MPTCP es como funciona, en concreto como envía datos y como gestiona las ventanas de congestión, además de conocer como establece los sub-flujos, como elige el sub-flujo que usara.

Lo primero que hace el protocolo, una vez establecida la conexión es "investigar" que otros posibles caminos existen, esto es, inicializar los sub-flujos. Esta labor se realiza mediante la función **initiateSubflows()**. La función barre todas las posibles combinaciones de direcciones IP de la siguiente forma, buscando las interfaces que no se hayan usado ya, pues como se mencionó en el capítulo anterior, dos direcciones IP que ya hayan establecido un sub-lujo no podrán ser usadas para otros. La búsqueda de nuevos sub-flujos es:

```
(for i= 0; i < localAddress; i++)
        for(j = 0; j < remoteAddress; j++)
            //Se comprueba si es posible establecer un sub-flujo entre i y j.
```

De esta manera, se establecen los sub-lujos que se emplearan durante la transmisión. También es posible añadir o eliminar sub-lujos de forma manual, mediante **Connect()** o **Close().**

El siguiente punto importante que interesa es conocer cómo se envía un paquete. Esta tarea no puede ser abordada sin implicar al control de congestión y el *scheduler*, para conocer una cosa se debe explicar cómo funciona la otra. El funcionamiento es el siguiente:

Para enviar un paquete, se llama a la función **SendDataPacket()**, quien, preparara el paquete y hará una llamada a **setReTxTimeout()**, que establece un tiempo máximo de espera, y una vez concluido, retransmite el paquete, siendo necesaria la recepción de la confirmación del paquete de datos antes de que ese tiempo expire. Una vez establecidos los diferentes parámetros, queda conocer por donde se enviarán los datos y para ello se llama a la función **getSubflowToUse()**, función que se vera de nuevo en la siguiente sección, pues se tuvo que modificar para adaptar a las necesidades del trabajo. La función obtendrá la ID del sub-flujo por el cual enviar los datos, en consecuencia, del *shceduler* que se haya establecido. Una vez conocido el sub-flujo a usar, se envían los datos.

Es en este momento cuando el receptor puede o no recibir el paquete y, por consiguiente, enviar o no la confirmación del mismo. Ambas situaciones provocaran en el transmisor una modificación de la ventana de congestión, de acuerdo a lo que dicte el control de congestión.

Si se produce la confirmación, y esta llega sin problemas al receptor, en este se produce una llamada a **NewACK()** quien pospone la retransmisión establecida. Realmente un ACK dispara la siguiente cadena de eventos:

**ReceivedACK() → NewACKNewReno() → NewACK()**

Siendo **NewACK()** quien se encarga de modificar la ventana de congestión, mediante una llamada a la función **OpenCWND()**, función que, de nuevo, será tratada en la próxima sección, debido a que fue modificada.

**OpenCWND()** modificara la ventana de congestión, basándose en qué control de congestión, **AlgoCC**, este fijado, volviendo al momento en el que se debe llamar a **SendDataPacket()**.

Por otro lado, si no se envía la confirmación, esta no llega, o el paquete de datos se ha perdido por la red, es el transmisor quien debe ejecutar la retransmisión, pues estaba programada para un tiempo máximo, que habrá expirado. Cuando el tiempo llega a su fin (o, mejor dicho, llega la hora para la que estaba establecida la retransmisión), el transmisor llama a **ReTxTimeout()** llamando a su vez a **Retransmit()**, quien, de nuevo, será la entidad que reduzca la ventana de congestión, gracias a la función **ReduceCWND()**, para después, hacer una llamada a **DoRetransmit()**. Por último, esta función lo único que hace es enviar el paquete (retransmisión) correspondiente.

Es de esta manera cómo funciona el envió de datos, el control de congestión y *scheduler* en MPTCP. Falta añadir que en un principio todas estas funciones iban a ser duplicadas, crean así una alternativa con la funcionalidad deseada dejando la original intacta, pero finalmente se decidió implementar la funcionalidad buscada junto a la ya existente (dando opción a elegir cual emplear mediante atributos de la clase), que se verá en la siguiente sección.

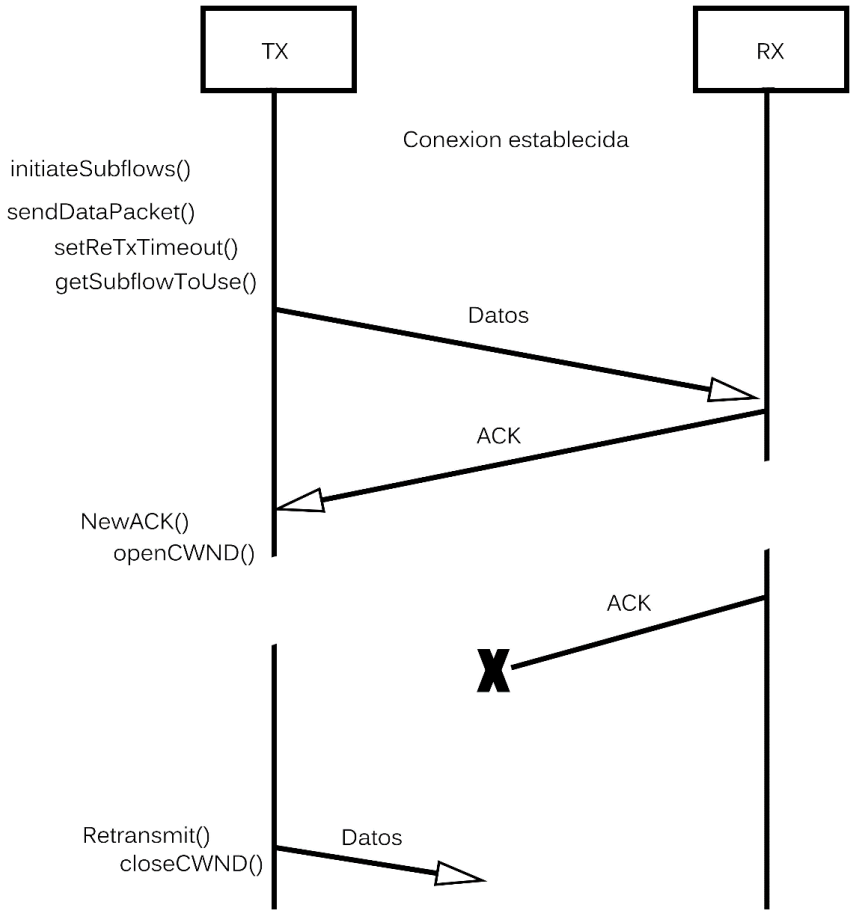


*Figura 3.6 – Esquema de transmisión MPTCP*

En la figura 3.6 se muestra un esquema de la funcionalidad descrita, en la que se pueden apreciar todos los procesos ya mencionados.

Por otro lado (de hecho, se encuentra en un fichero del código fuente totalmente diferente) se encuentra la aplicación que se ha desarrollado para MPTCP. Un transmisor y un receptor que funcionan sobre este protocolo y que son necesarios para llevar a cabo las simulaciones.

Se trata de **MpTcpBulkSendHelpe**r y de **MpTcpPacketSinkHelper**. Dos clases que se instalan como aplicación en un nodo, transmisor y receptor, respectivamente y tras darles una dirección de destino (en el caso del transmisor) y un tiempo a partir del actual se encontrar activos, el transmisor comenzara a enviar paquetes de datos y el receptor estará escuchando el medio y recibiendo los paquetes, enviando la confirmación pertinente cuando sea necesario.

A pesar de ser aplicaciones básicas, se adaptan perfectamente a las necesidades del trabajo, donde lo que hace falta es un transmisor que emita datos con el fin de estudiar la evolución de las ventanas de congestión.

## 3.5 –Adaptación del *Scheduler* y Control de Congestión

Como se ha mencionado, el código para la funcionalidad del control de congestión y del *scheduler* fue implementado desde cero, para soportar las necesidades del trabajo. Tras un primer proceso de comprensión de ambos mecanismos, y tras localizar las funciones del código que lo realizaban, queda adaptar los algoritmos al código, empleando los recursos disponibles del módulo.

### 3.5.1 – Control de congestión

Al control de congestión se accede mediante las funciones **reduceCWND()** y **openCWND()**. Ambas funciones tienen implementadas todas las opciones de los distintos algoritmos que se vieron en el capítulo anterior, todas ellas bajo un **swItch()** que seleccionara cual emplear en función del atributo **MpTcpSocketBase::SetCongestionCtrlAlgo** y **AlgoCC**.

Los dos algoritmos más complejos, Linked Increases y RTT Compensator hacen uso de las funciones **compute_alpha()** y **calculateAlpha()**, que calculan el valor $\alpha$ , valor que representa el factor de agresividad y que se calcula con su expresión correspondiente.

Todos los algoritmos emplean la función **compute_total_window()** la cual, simplemente, recorre todos los sub-flujos existentes tomando el valor de sus ventanas y sumándolos.

### 3.5.2 – Scheduler

El caso del *Scheduler* es más complicado, pues el algoritmo que vimos en el capítulo 2 es mucho más complejo, ya que se basa en valores que no son tan fácilmente obtenibles en el simulador, siendo este el motivo por el cual se decidiese implementar un Schedule más sencillo.

Inicialmente el único *Scheduler* implementado era un simple Round Robin, que elegía el sub-flujo siguiente al último empleado. A lo largo del trabajo se implementó un primer algoritmo, trivial, que emplea el primer sub-flujo, independientemente de la cantidad de sub-flujos establecidos. Este *Scheduler* no tiene mayor valor que el valor experimental cuya finalidad es depurar código.

El *scheduler* implementado elegiría el sub-flujo basándose en la mayor ventana de congestión individual (ya se vio que cada sub-flujo tiene una ventana independiente del resto), pues ese sub-flujo será el que menor congestión presente y por tanto el más adecuado. El algoritmo se desarrolló para un uso de dos sub-flujos como máximo, quedaría desarrollar una implementación para un uso de *n* sub-flujos. El algoritmo se limita a recorrer los sub-flujos mirando y comprando entre ellos en busca de la mayor ventana. Para el caso de *n* sub-lujos, la búsqueda sería algo más compleja.

# 4

# Resultados obtenidos

En este capítulo se exponen todos los resultados obtenidos de las diferentes simulaciones que se han realizado, para ello, primero se exponen las formas por las cuales los datos han sido obtenidos a partir del código, así como una explicación de los datos que se están obteniendo. Después se explicará el proceso que se ha seguido para obtener los resultados para, a continuación, explicar los escenarios que se han simulado con sus diferentes características. Por último, se presentarán y describirán los resultados de cada uno de los escenarios simulados.

## 4.1 – Obtención de resultados

El objetivo de la simulación es observar el comportamiento y el rendimiento de los diferentes algoritmos de control de congestión en varios escenarios.

Antes de hablar sobre la forma en la que se ha manipulado del código para obtener los datos, primero se darán a conocer los datos que se han obtenido.

- ***Throughput***: se trata del cociente entre los datos recibidos correctamente y el tiempo que ha durado la transmisión de los mismos. Se trata de un parámetro fundamental en cualquier estudio ya que da una idea sobre el rendimiento de la red.

$$Throughput = \frac{Total\ bits}{Tiempo\ simulación}$$

- **Ventana de congestión**: como ya se ha definido anteriormente, la ventana de congestión (*cwnd*) indica al transmisor el número de bytes que puede tener "en vuelo", es decir, que puede haber enviado sin haber recibido confirmación.

La ventana de congestión en MPTCP es independiente de cada sub--flujo, y por tanto cada una variara en función de la congestión de su camino. Inicialmente a la ventana se le asigna un valor de 1 MSS (*Maximum Segment Size*), y a partir de dicho valor, aumentará o disminuirá en función del algoritmo de congestión, que se ha definido en el capítulo 2.

Este parámetro nos da una idea de lo congestionado o no que puede estar un flujo.

- **Estado de las ventanas de congestión**: adicionalmente al punto anterior, y como parámetro intrínsecamente ligado a un escenario *mulitpath*, el estado de la ventana combina el valor de las diferentes ventanas para cada paquete de datos transmitido, así como el sub--flujo por el cual se ha enviado el último paquete de datos y el tiempo que ha pasado hasta que ha vuelto a enviar otro paquete.

  Este último valor nos permitirá saber el tiempo que ha pasado en cada posible combinación de valores de ventanas congestión.

Conociendo ya la definición de los datos que se han obtenido, a continuación, se explicara el proceso por el cual se han obtenido, para una única simulación del escenario.

- ***Throughput***: este parámetro se ha medido en el receptor de manera sencilla. Cuando se inicia la transmisión se toma una referencia de tiempo, a partir de ahí solo queda contar el número de bits que llegan a la aplicación. Por último, cuando se finaliza la transmisión se vuelve a tomar una referencia temporal. De esta manera, al finalizar la simulación, la clase hace el cálculo y se devuelve al usuario.

- **Ventana de congestión**: respecto al anterior, este parámetro es más difícil de obtener, pues no se puede hacer un cálculo en el transmisor, que lo único que recibe son paquetes. Para la correcta medida del valor, hay que profundizar más en el código y llegar al momento en el que se comparan las ventanas de congestión. Como se vio en el capítulo 3, esto se produce en la función **getSubflowToUse ()**, quien examina todas las ventanas de congestión en busca de la mayor (en el caso el algoritmo implementado). Sera aquí donde, una vez se realice esta comparación, se obtenga el valor de las ventanas, que se irán guardando en un objeto de clase *map*.

- **Estado de la ventana de congestión**: aprovechando la lectura del anterior parámetro, medir el tiempo y el flujo por el cual se enviará el paquete resulta sencillo. El valor del sub-flujo por el que se enviara el paquete de datos es el propio resultado de la función. El tiempo que el transmisor permanece en ese estado se calcula como una diferencia de tiempos que se reinicia cada vez que se envía un nuevo paquete, almacenando el valor anterior en una variable.

Obtenidos esta serie de valores para cada simulación, solo queda devolverlos al usuario. En este caso, esta tarea se lleva a cabo mediante la escritura de todos los valores obtenidos un fichero de salida para cada simulación.

## 4.2 – Proceso de medida

Cuando es hora de obtener los resultados de manera consistente, una única simulación del escenario no es suficiente. Es por ello que, para cada escenario a simular, se han realizado varias simulaciones, posteriormente se ha promediado los valores obteniendo así resultados más estables.

Para llevar a cabo esta operación, se ha llamado a la simulación como si fuera una función más, dentro de un bucle que llamara a la función ***N*** veces, guardando los resultados de cada ejecución de la función y promediándolo después.

Al realizar este proceso, surge un inconveniente, con una solución fácil, pero que hay que destacar. Resulta que cada ejecución de la simulación, si no se aplica ninguna solución, proporcionara exactamente los mismos datos que la anterior, resultado, por tanto, la misma simulación que la anterior, arruinando así el proceso de medida. La solución es tan sencilla como aplicar un *random seed* a cada ejecución de la simulación, haciendo de esta manera que cada simulación sea independiente y diferente a las otras.

Por último, hay que añadir que el valor de las ventanas de congestión en cada instante no se puede promediar, pues el resultado que obtenemos no sería el deseado si se pretende observar como ha actuado el algoritmo de control de congestión. Por ello los valores de las ventanas de congestión de cada simulación se almacenarán para posteriormente cuantificar cual es o son los estados más frecuentes, y cómo se comporta ante determinada situación.

Para cada escenario se eligió un numero de simulaciones entre 20 y 25, dependiendo del tiempo de ejecución en cada caso, así como una duración de 40 segundos por simulación.

Las gráficas se han obtenido mediante MATLAB.

## 4.3 – Descripción de escenarios

En esta sección se verá con detalle cada escenario, aportando un esquema de la topología del mismo. Cabe añadir que a lo largo del desarrollo del trabajo se han implementado más escenarios que los aquí presentes (y más sencillos) cuyo único objetivo era la depuración del módulo MPTCP y no la medida de rendimiento, haciendo esto que no se tomasen valores de estos escenarios de prueba. Se verá que uno de los escenarios aquí expuestos es el escenario básico para una topología multicamino. A pesar de que no es el objetivo del trabajo, se ha añadido en esta sección como un ejemplo más.

### 4.3.1 – Escenario 1

Se trata del escenario clásico, que ya se vio en la figura 2.1 y como se dijo, es el escenario básico a la hora de realizar experimentos en un sistema multicamino. Este escenario está compuesto de 4 nodos conectados con enlaces cableados.

El escenario posee las siguientes características:

- Segment Size: 1400
- Máximo número de Sub--flujos: 2
- Algoritmo del *Scheduler*: Máxima ventana de congestión (MAX_CWND)
- Algoritmo de Control de congestión:
  - UNCOUPLED
  - FULLY COUPLED
  - LINKED INCREASES
- Probabilidad de error de las conexiones:
  - 0.00001 y 0.00001
  - 0.00001 y 0.00005
  - 0.00009 y 0.00005
- Tiempo de simulación: 40 segundos
- Numero de simulaciones: 25

En esta topología el direccionamiento IP se genera automáticamente mediante la llamada a *Ipv4GlobalRoutingHelper::PopulateRoutingTables()*, completándose las tablas de enrutamiento automáticamente.

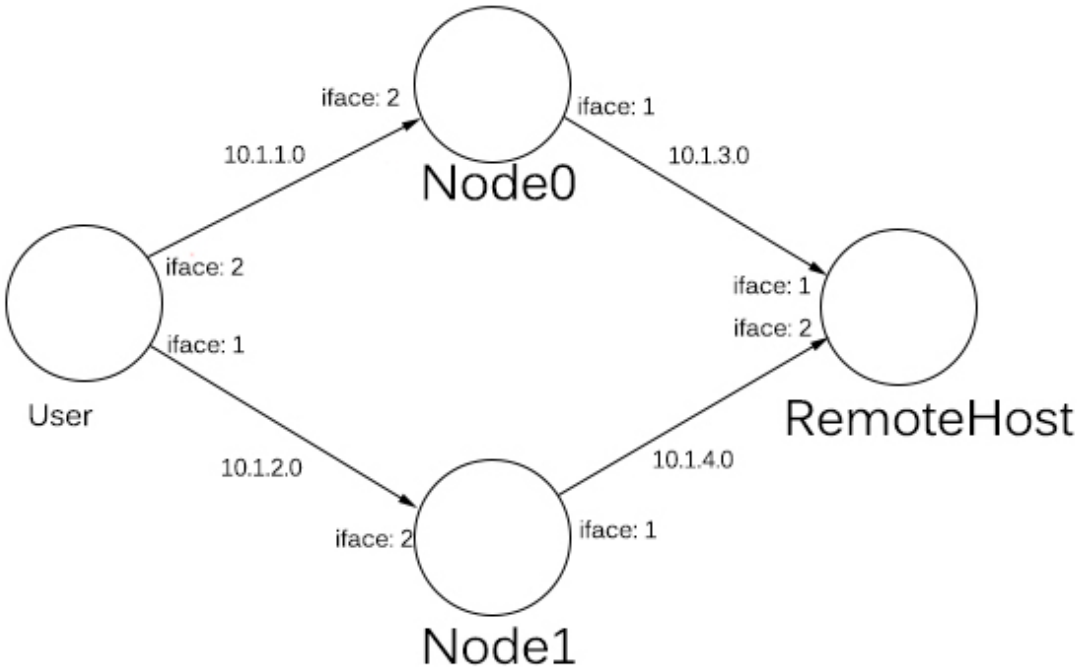


*Figura 4.1 – Topología y detalles del escenario 1*

Las probabilidades de error y el modelo de error se aplican en los enlaces entre los nodos:

- PE0: User → Node0
- PE1: User → Node1

El escenario no presenta mayor complejidad, pero servirá para estudiar la respuesta de los algoritmos de control de congestión.

### 4.3.2 – Escenario 2

En este caso se trata de un escenario en la que un terminal está conectado a dos redes wifi. Esto se logra instalando dos dispositivos wifi en el mismo nodo. En este escenario es fundamental emplazar correctamente los nodos mediante el modelo de movilidad, pues en caso de colocarlos en posiciones incorrectas, la simulación no será válida, pudiendo, incluso, terminar con algún error y no llegar a ejecutarse completamente.

Otro problema que presenta esta topología es la generación de tablas de rutas en cada nodo. El comando que se vio en el caso anterior no produce buenos resultados para el caso de redes inalámbricas, siendo entonces necesario generar las tablas de alguna manera. En un escenario de tan pocos nodos se eligió generarlas de manera manual, rellenando las tablas con las opciones para alcanzar de forma correcta a los destinos.

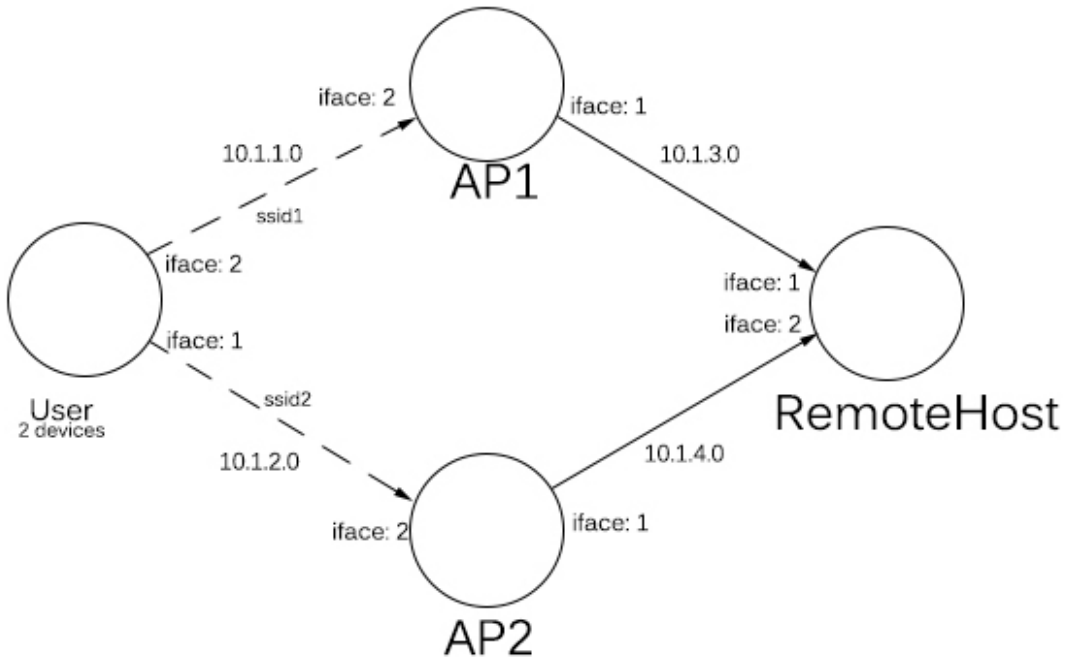


*Figura 4.2 – Topología y detalles del escenario 2*

Las características del escenario son:

- Segment Size: 1400
- Máximo número de Sub-flujos: 2
- Algoritmo del *Scheduler*: Máxima ventana de congestión (MAX_CWND)
- Algoritmo de Control de congestión:
  - UNCOUPLED
  - FULLY COUPLED
  - LINKED INCREASES
- Probabilidad de error de las conexiones:
  - 0.00001 y 0.00001
  - 0.00001 y 0.00005
  - 0.00009 y 0.00005
- Tiempo de simulación: 40 segundos

- Numero de simulaciones: 25

### 4.3.3 - Escenario 3

Se trata del escenario más complejo de todos los estudiados. Hace uso del módulo LTE de ns-3. El desarrollo de este escenario presento los siguientes inconvenientes.

De nuevo es fundamental la ubicación de los nodos de acceso wifi, del eNB y del nodo de usuario, produciendo errores en caso de un mal posicionamiento. Y también se produce el mismo percance que en el escenario 2 con las tablas de direccionamiento, optando otra vez por la opción de rellenarlas manualmente.

A pesar de incluir el módulo LTE, mucho más complejo que el lado wifi, este se gestiona de manera automática y cerrada, produciendo los resultados esperados. El propio modulo se encarga del direccionamiento de las interfaces del UE, así como la gestión de toda la parte del EPC.

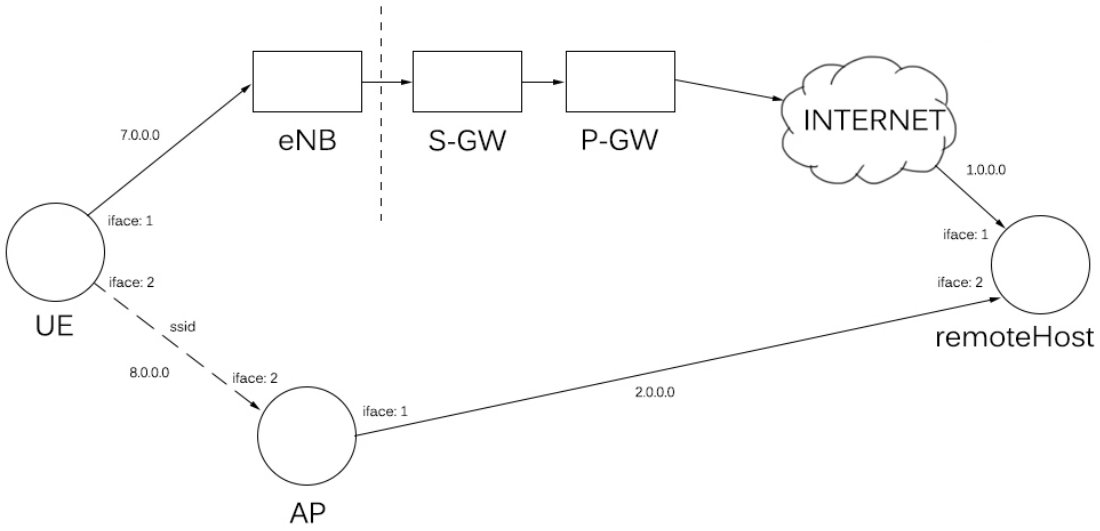


*Figura 4.3 – Topología y detalles del escenario 3*

Un nuevo inconveniente que surge en este escenario es la generación de probabilidades de error en los enlaces LTE. Al no estar modelado, la generación de errores se ha producido de una forma más "rudimentaria". Se ha realizado un estudio previo en el que se establece la distancia entre UE y eNB, y la probabilidad de error que eso conlleva, alejando cada vez más el UE, la probabilidad de error aumenta, estableciendo así una relación directa.

El escenario posee las siguientes características:

- Segment Size: 1400
- Maximo numero de Sub-flujos: 2
- Algoritmo del *Scheduler*: Máxima ventana de congestión (MAX_CWND)
- Algoritmo de Control de congestión:
  - UNCOUPLED
  - FULLY COUPLED
  - LINKED INCREASES
- Probabilidad de error de las conexiones:

    - 0.00001 y 0.00001
    - 0.00001 y 0.00005
    - 0.00009 y 0.00005
- Tiempo de simulación: 40 segundos
- Numero de simulaciones: 25

## 4.4 – Resultados obtenidos

En esta última sección se exponen y se comentan los resultados obtenidos en los tres escenarios ya descritos. Para cada escenario se ha llevado a cabo una serie de simulaciones con diferentes valores de probabilidad de error en cada enlace y diferentes algoritmos de control de congestión.

Cabe destacar que el estudio del comportamiento de las ventanas de congestión se ha llevado a cabo por igual en todos los escenarios, obteniendo resultados similares (dentro de lo que cabe, pues se trata de distintas simulaciones que presentan pequeñas variaciones de datos), es por este motivo que la presentación de gráficas y razonamientos sobre los resultados solo se plasmaran en el apartado correspondiente al escenario 1, evitando así repetir graficas que muestren prácticamente los mismos datos y conduzcan a las mismas conclusiones. Por otro lado, los escenarios 2 y 3 tienen mayor relevancia en el estudio del *throughput*.

### 4.4.1 – Escenario 1

Este escenario se estudia como punto de partida y referencia para los posteriores resultados.

Al iniciar la simulación, se establecen los dos sub-flujos con los que trabajara la conexión automáticamente. En la figura 4.4 se pueden ver los dos diferentes que se han establecido, cada uno en un color. A partir de aquí, se irán cambiando las características del escenario (probabilidad de error, algoritmos…) y viendo cómo reaccionan a cada caso.

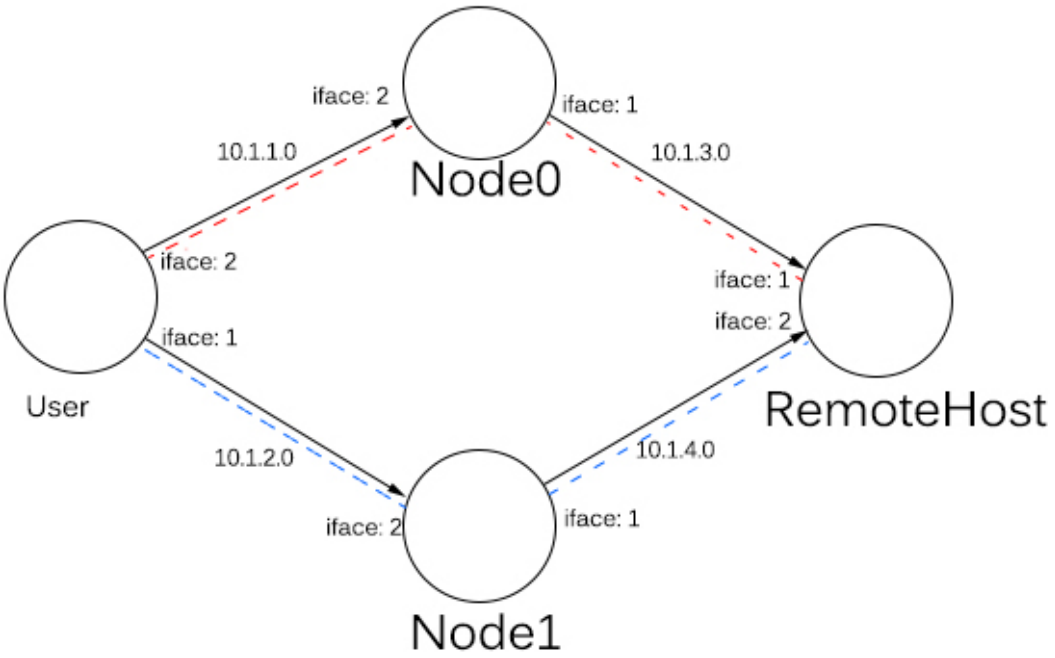


*Figura 4.4 – Sub--flujos establecidos en el escenario 1*

#### 4.4.1.1 – Caso 1

Se emplea un algoritmo UNCOUPLED, es decir, las ventanas de congestión son totalmente independientes entre ellas.

Esta configuración presenta las siguientes características:

- Algoritmo de scheduling: MAX_CWND (Mayor ventana de congestion)
- Algoritmo de control de congestion: UNCOUPLED
- Probabilidad de error en los enlaces: 0.00001 en el sub-flujo 1 y 0.00001 en el sub-flujo 2

En la figura 4.5 se presentan, por un lado, el porcentaje de uso de cada flujo. Para este caso cada sub-flujo se usa el mismo número de veces, exactamente lo que cabe esperar, ante dos enlaces con las mismas características se emplearán uno u otro independientemente.

La grafica de la izquierda muestra las ventanas de congestión en MSS, donde cada punto es un estado de las dos ventanas de congestión. Se pueden diferencias tres zonas; la primera, la gran "nube" de puntos en la zona central que se representa en la gráfica como una mancha. Esta zona se corresponde con el uso indefinido de cualquiera de los dos caminos, es decir, se elegirá un flujo, se enviará por ahí hasta que se produzca un error, se disminuirá la ventana y muy posiblemente el valor de esta quedara por debajo del otro, por lo que el *scheduler* elegirá el otro sub-flujo.

Las otras dos zonas son los "brazos" que se prolongan de la nube central. Se puede apreciar que son pocas las muestras en estos brazos en comparación con la zona central. Estos puntos se interpretan como momentos en los que no se produce ningún error en una de las ventanas,

dejando que esta aumente considerablemente, lo que implica que cuando se produce un error, la ventana se reduce, pero no lo suficiente como para utilizar el otro camino.

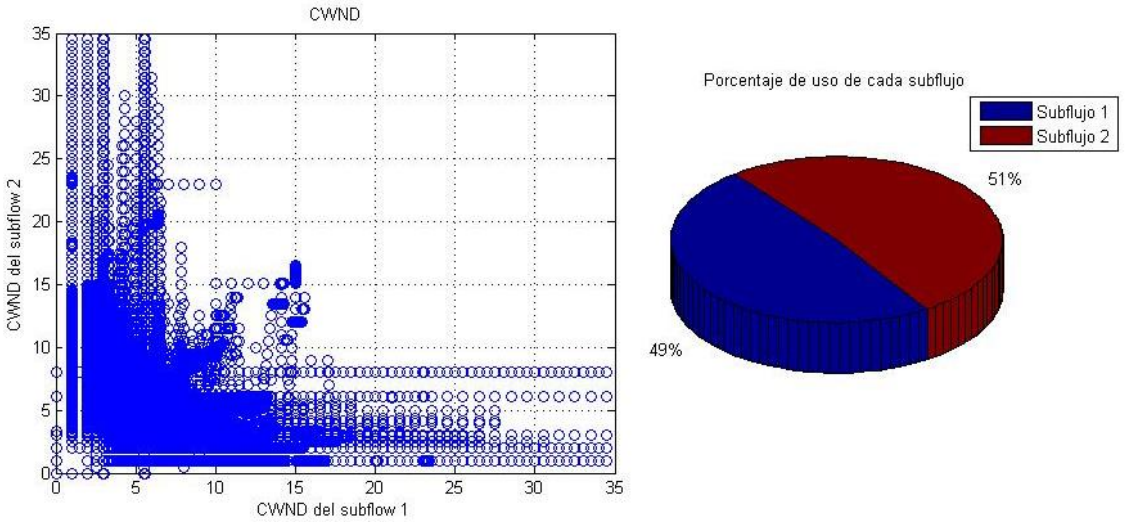


*Figura 4.5 – Uso de ventanas de congestión y porcentaje de uso de cada flujo para el caso 1 en igualdad de probabilidad de error*

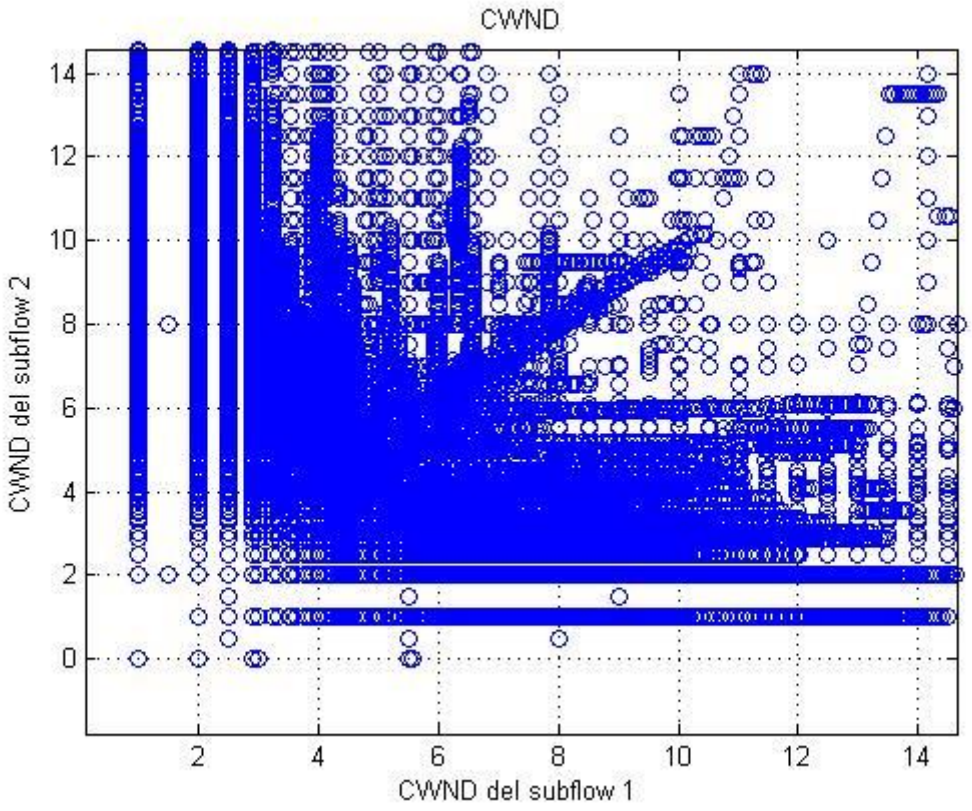


*Figura 4.6 – Detalle de las ventanas de congestión para el caso 1*

Una nueva configuración, variando las probabilidades de error de los enlaces para ver cómo reacciona MPTCP.

- Probabilidad de error en los enlaces: 0.00001 en el sub-flujo 1 y 0.00005 en el sub-flujo 2

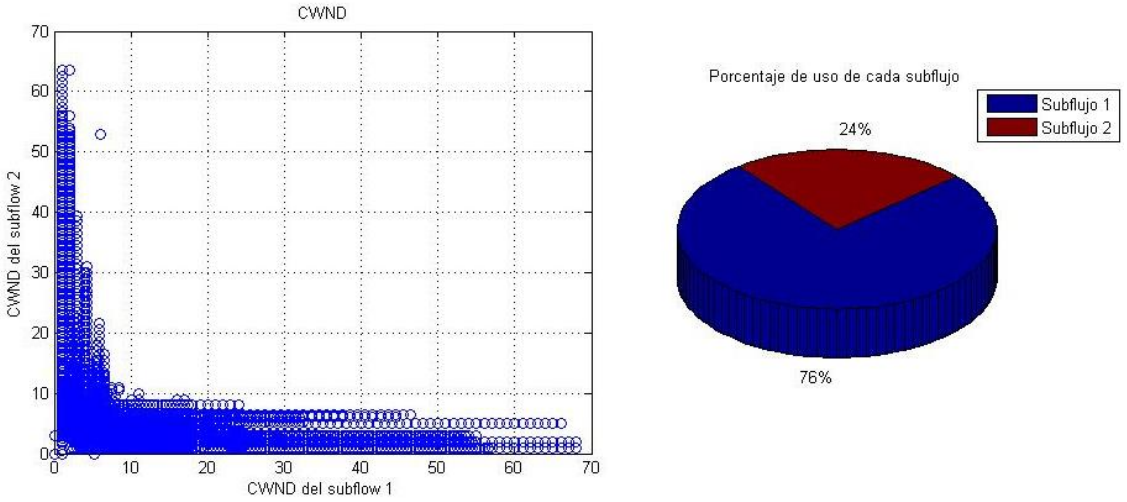


*Figura 4.7 – Uso de ventanas de congestión y porcentaje de uso de cada flujo para el caso 1 con diferentes probabilidades de error*

En esta ocasión los enlaces no son simétricos, lo que implica que el *scheduler* tienda a emplear el que produce menos errores, siendo el sub-flujo 1 el preferido. En la gráfica de la izquierda, la "nube" central que se mencionaba para la anterior situación, parece que se inclina y se alarga hacia la derecha, o, mejor dicho, hacia el sub-flujo 1, por esta misma causa.

Este algoritmo parece funcionar como se deseaba, pero quizás siga dando demasiada importancia al uso del enlace con mayor error, aparte de producir ese efecto indeseado que ya hemos comentado.

#### 4.4.1.2 – Caso 2

El algoritmo a emplear será el FULLY COUPLED, con intención de reducir el "*flappiness*" que ya se ha mencionado. En este caso las ventanas de congestión si estarán vinculadas entre ellas, y una perdida en un sub-flujo implicara cambios en ambos.

Se cambia el algoritmo de control de congestión. Ambos enlaces se encuentran en el mismo estado.

Esta configuración presenta las siguientes características:

- Algoritmo de *scheduling*: MAX_CWND (Mayor ventana de congestión)
- Algoritmo de control de congestion: FULLY COUPLED
- Probabilidad de error en los enlaces: 0.00001 en el sub-flujo 1 y 0.00001 en el sub-flujo 2

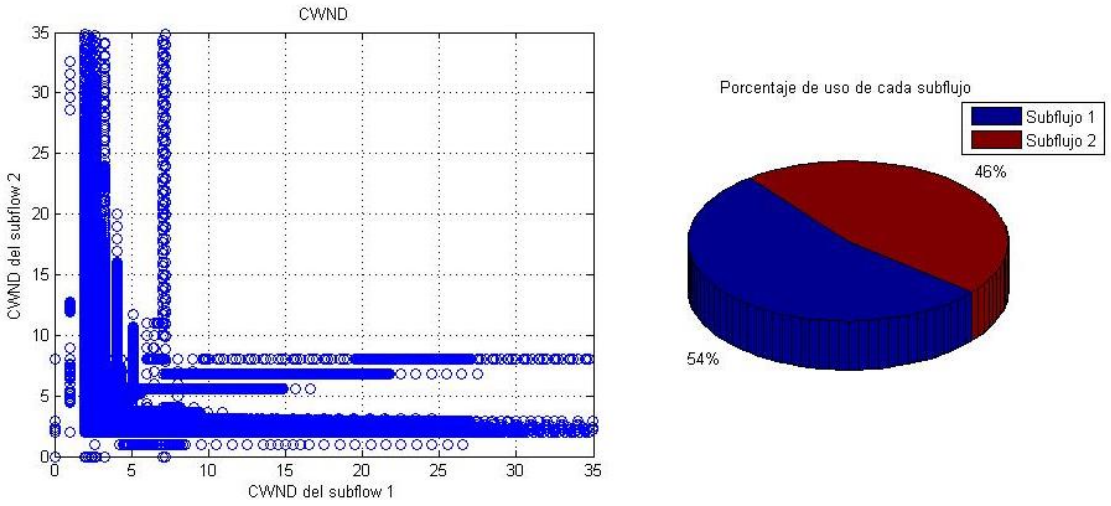


*Figura 4.8 – Uso de ventanas de congestión y porcentaje de uso de cada flujo para el caso 2 en igualdad de probabilidad de error*

Y tal como se pretendía, el algoritmo reduce ese efecto de "*flappiness*". De nuevo el uso de cada camino es igual, no hay ninguno que predomine sobre otro. En la gráfica de las ventanas de congestión ahora solo hay dos zonas diferenciadas; los dos "brazos". Esto quiere decir que el algoritmo empleará un camino u otro, pero no habrá esa zona intermedia que se veía en el caso 1. En otras palabras, tras transmitir por un sub-flujo se continuará por el hasta que los errores sean suficientemente notables, en ese momento se elegirá el otro, se podría decir que funciona como un interruptor, llevando toda la carga por un flujo o por el otro, pero no repartiéndolo. Esto se verá mejor cuando uno de los sub-flujos sea mejor que otro.

Se cambiará la configuración, aumentando la probabilidad de error en uno de los enlaces:

- Probabilidad de error en los enlaces: 0.00001 en el sub-flujo 1 y 0.00005 en el sub-flujo 2.

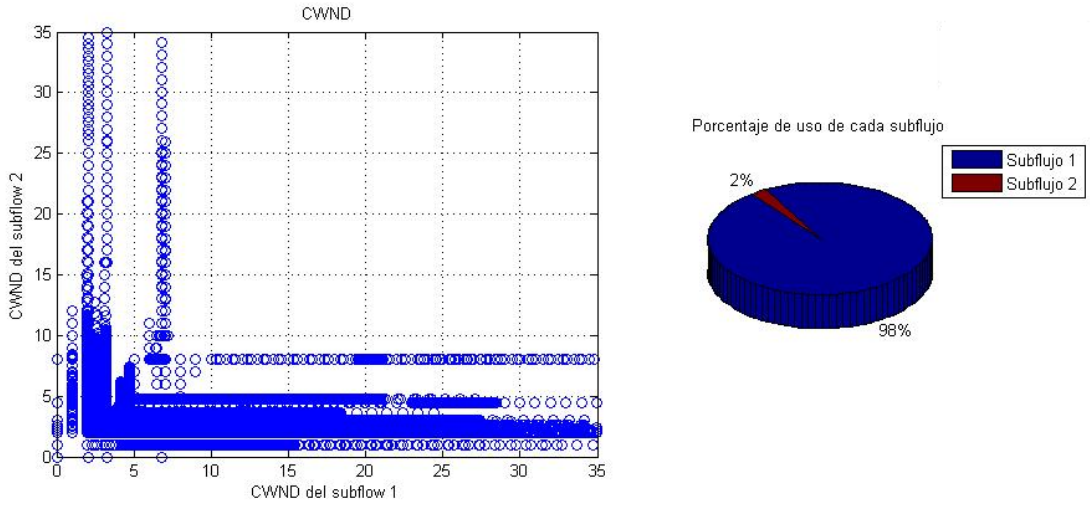


*Figura 4.9 – Uso de ventanas de congestión y porcentaje de uso de cada flujo para el caso 2 con un flujo predominante*

Tal y como se había dicho, este caso dispara ese comportamiento de interruptor en el que se envía toda la carga por un único camino. La grafica muestra claramente como la única ventana que se usa es la correspondiente al sub-flujo 1.

Se puede concluir que este algoritmo reduce el "*flappiness*" pero a cambio aporta este comportamiento peculiar. Comportamiento que puede ser interesante dependiendo de la aplicación a usar.

Al desbalancear algo, sin llegar al extremo de esta configuración, las probabilidades de error:

- Probabilidad de error en los enlaces: 0.00010 en el sub-flujo 1 y 0.00005 en el sub-flujo 2.

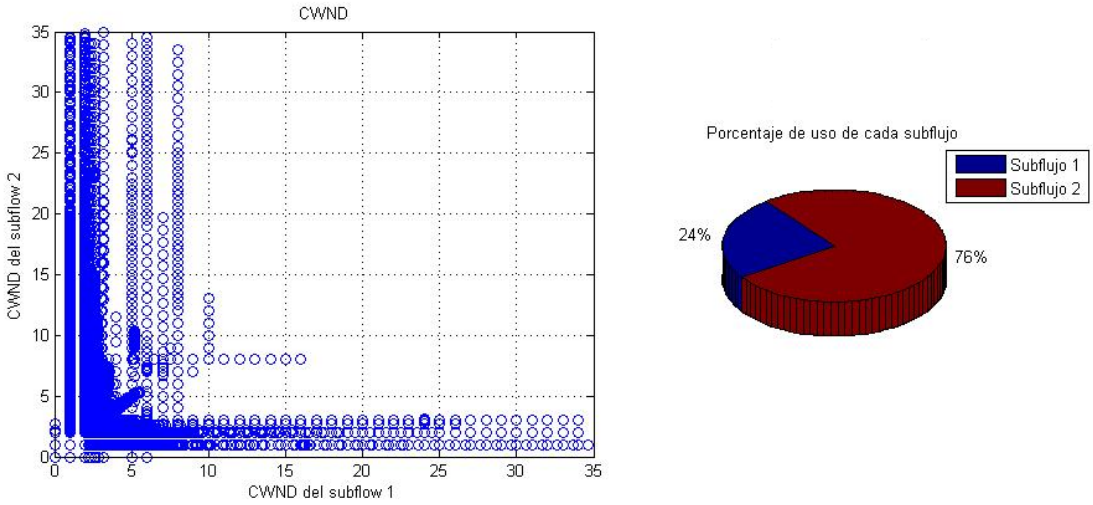


*Figura 4.10 – Uso de ventanas de congestión y porcentaje de uso de cada flujo para el caso 2 con cierto balance de carga*

Esta última configuración, aun balanceando algo la carga, se sigue comportando de la misma manera. En la gráfica de la izquierda se sigue usando mayoritariamente la ventana del sub-flujo predominante. Es cierto que en esta ocasión se puede ver como la otra ventana gana algo de protagonismo, hasta que se produce un error en la transmisión, momento en el que el *scheduler* cambia al otro sub-flujo.

#### 4.4.1.3 – Caso 3

Por último, se sustituirá el algoritmo por el LINKED INCREASES. Ya se vio en el capítulo 2 como funcionaba el algoritmo y que metas quiere cumplir. Una primera configuración donde los enlaces presentan las mismas características mostraran el comportamiento del algoritmo:

- Algoritmo de *scheduling*: MAX_CWND (Mayor ventana de congestión)
- Algoritmo de control de congestión: LINKED INCREASES
- Probabilidad de error en los enlaces: 0.00001 en el sub-flujo 1 y 0.00001 en el sub-flujo 2

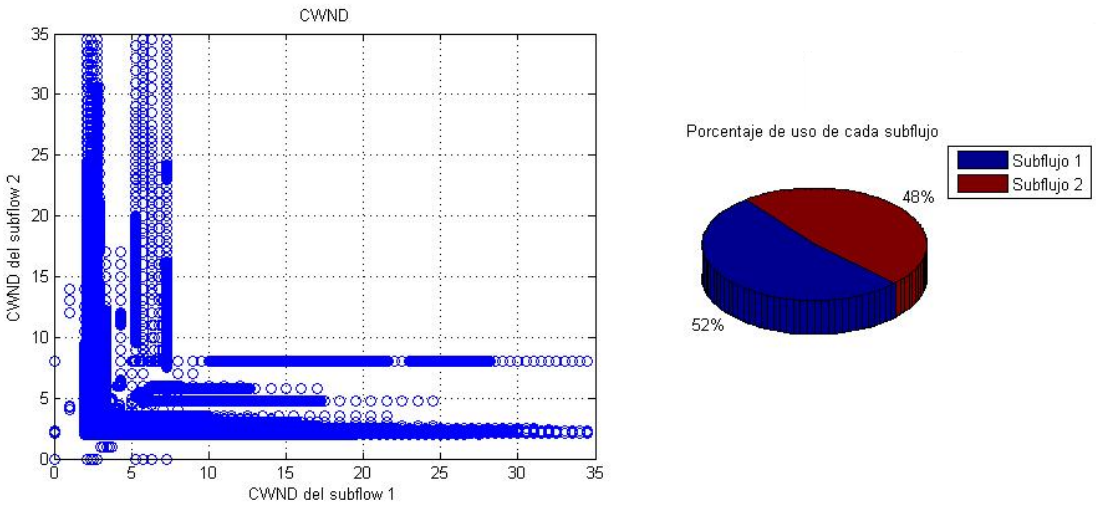


*Figura 4.11 – Uso de ventanas de congestión y porcentaje de uso de cada flujo para el caso 3 en igualdad de probabilidades de error*

LINKED INCREASES tiene como objetivo reducir el "*flappiness*" al igual que el caso anterior, pero actuando de una forma diferente. Con este esquema de control de congestión no hay un único "brazo" en la gráfica de las *cwnd*, se pueden apreciar varios segmentos, es decir, las ventanas se mueven entre estos segmentos determinados, de forma discreta. En esta configuración el reparto de la carga está equilibrado. En otras palabras, este algoritmo está a medias entre los dos anteriores, actúa como un interruptor, pero no tendrá dos únicas opciones que representen el todo o nada, sino que el interruptor podrá decidir entre un número limitado de opciones, balanceando más o menos la carga. Tampoco está en el extremo del UNCOUPLED en el que él las opciones que puede tomar las ventanas de congestión son infinitas.

Cambiando la probabilidad de error se aprecia mejor este comportamiento:

- Probabilidad de error en los enlaces: 0.00001 en el sub-flujo 1 y 0.00005 en el sub-flujo 2

En un extremo en el que los dos flujos tienen diferencias notables, el algoritmo se comporta de manera similar al FULLY COUPLED, dando a elegir una única opción para enviar los datos. Si es cierto, que se ven otros segmentos en la gráfica.

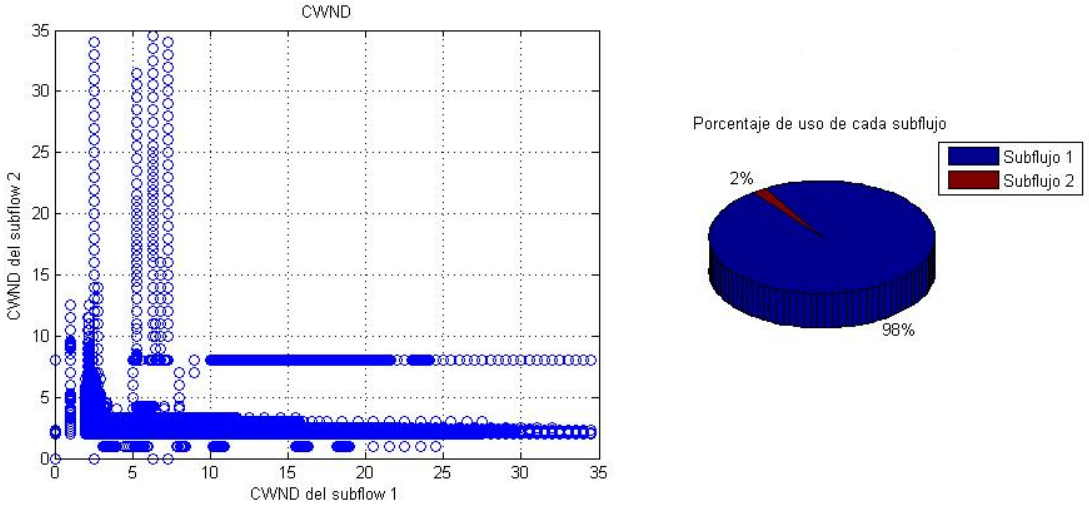


*Figura 4.12 – Uso de ventanas de congestión y porcentaje de uso de cada flujo para el caso 3 con un flujo predominante*

Por último, se aplica una configuración más equilibrada:

- Probabilidad de error en los enlaces: 0.00005 en el sub-flujo 1 y 0.00010 en el sub-flujo 2

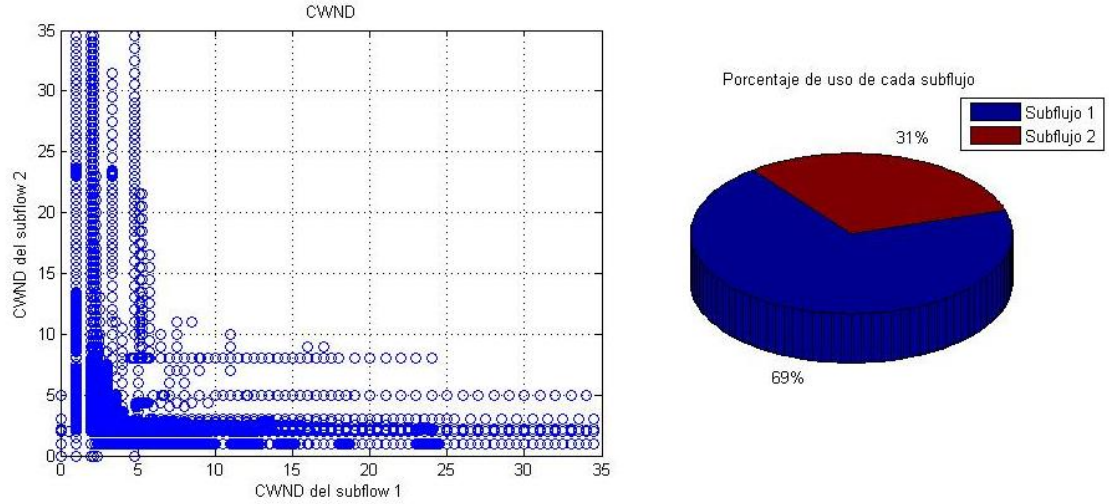


*Figura 4.13 – Uso de ventanas de congestión y porcentaje de uso de cada flujo para el caso 3 con cierto balance de carga*

Este caso muestra una distribución de carga más lógica a la del caso anterior, aproximándose de nuevo al caso FULLY COUPLED.

### 4.4.2 – Escenario 2

El escenario 2 hace uso de dos conexiones wifi a las que está conectado el nodo de usuario. Una vez iniciada la simulación, lo primero que se establece son los diferentes sub-flujos, dos en este caso (uno rojo y otro azul en la figura 4.4).

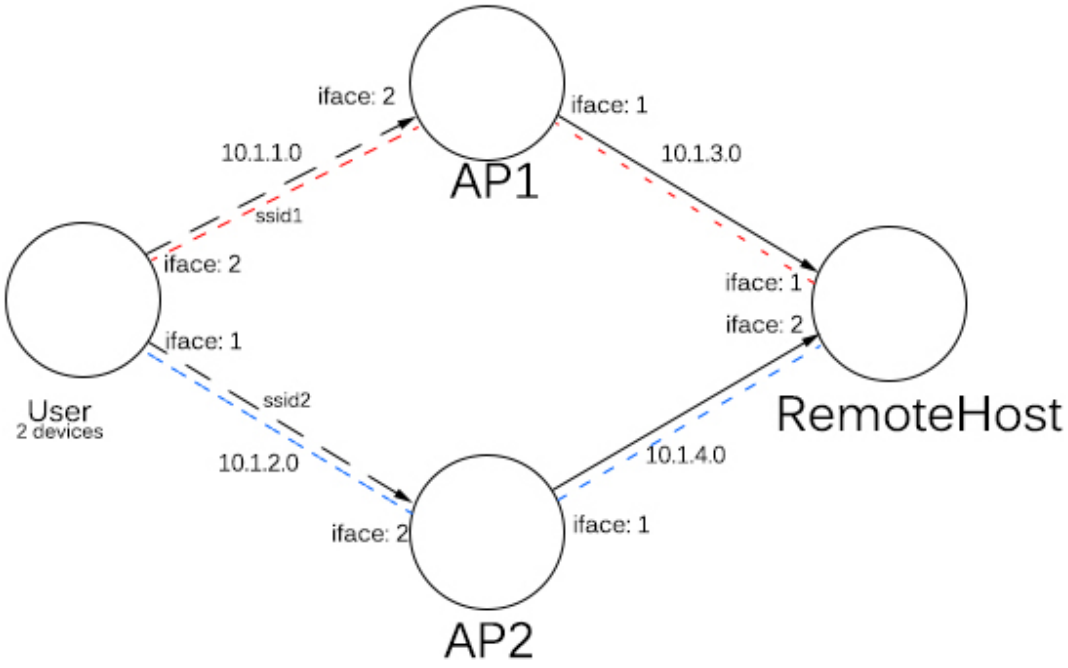


*Figura 4.14 – Sub--flujos establecidos en el escenario 2*

Para este escenario se han reproducido las mismas configuraciones que en el escenario 1, dando graficas similares, pues de cara al algoritmo de control de congestión la diferencia de topología no implica gran cosa. No es así en el caso del *throughput*, que veremos a continuación.

Otros estudios ya han demostrado le mejora de rendimiento de MPTCP frente a TCP tradicional en términos de *throughput* en diferentes escenarios. Este estudio está enfocado a determinar si los algoritmos a estudiar soportan bien la congestión en la red (si hacen bien su trabajo). Para ello tomaremos como referencia el rendimiento de la red para un caso en el que ambos flujos presenten un bajo nivel de probabilidad de error, observando que ocurre cuando se aumenta en los diferentes flujos.

Se tomará como valor de referencia el caso con las siguientes características:

- Algoritmo: UNCOUPLED
- Probabilidad de error: 0.00001 y 0.00001

El rendimiento representa el *throughput* en tanto porciento referenciado a este caso. El valor de referencia que se toma es el resultado obtenido del caso en el que se emplea el algoritmo de congestión UNCOUPLED y las probabilidades de error son mínimas e iguales. De esta forma se puede ver si disminuye el rendimiento, y en caso de que lo haga, que porcentaje. Si esta caída de rendimiento es grande significa que el algoritmo no procesa bien el aumento de la probabilidad de error en los *paths*.

*Tabla 4.1 – Resultados del escenario 2 – caso 1*

| Caso 1 – Algoritmo UNCOUPLED | | |
|---|---|---|
| **Prob. de error en sub-flujo 1** | **Prob. de error en sub-flujo 2** | **Rendimiento** |
| 0.00001 | 0.00001 | 100 % |
| 0.00005 | 0.00001 | 89.9 % |
| 0.00005 | 0.00002 | 87.3 % |
| 0.0005 | 0.0009 | 9 % |

*Tabla 4.2 – Resultados del escenario 2 – caso 2*

| Caso 2 – Algoritmo FULLY COUPLED | | |
|---|---|---|
| **Prob. de error en sub-flujo 1** | **Prob. de error en sub-flujo 2** | **Rendimiento** |
| 0.00001 | 0.00001 | 72.5 % |
| 0.00005 | 0.00001 | 70.8 % |
| 0.00005 | 0.00002 | 68.1 % |
| 0.0005 | 0.0009 | 9.7 % |

*Tabla 4.3 – Resultados del escenario 2 – caso 3*

| Caso 3 – Algoritmo LINKED INCREASES | | |
|---|---|---|
| **Prob. de error en sub-flujo 1** | **Prob. de error en sub-flujo 2** | **Rendimiento** |
| 0.00001 | 0.00001 | 68.5 % |
| 0.00005 | 0.00001 | 62.5 % |
| 0.00005 | 0.00002 | 62.5 % |
| 0.0005 | 0.0009 | 9.2 % |

En los tres casos se puede apreciar como el rendimiento se mantiene estable pese a la introducción de diferentes probabilidades de error en los dos sub-flujos, esto implica que la comunicación permanezca estable y tenga un comportamiento robusto ante los problemas que puedan surgir en la red. En todos los casos el rendimiento sufre una caída en picado cuando la probabilidad de error es muy grande.

De entre todos los datos el más representable es aquel en el que la probabilidad de error aumenta considerablemente en uno de los sub-flujos (puede representar una rotura de un enlacen o un estado de congestión puntual). Gracias a las gráficas vistas en la sección anterior se vio que el algoritmo sabe reaccionar, de distintos modos a estas situaciones, ahora, además, se puede añadir que esa reacción mantiene la velocidad de la conexión, a pesar de un incremento considerable de los errores.

De los tres algoritmos, el que mejor rendimiento ofrece es el primero, UNCOUPLED. Como ya vimos este algoritmo presenta un problema, y es el efecto de "*flappiness*". En estas simulaciones parece que no afecta, pero se vio en el capítulo 2 como pueden producirse situaciones en las que pueda reducir el rendimiento. Se puede decir que tanto FULLY COUPLED y LINKED INCREASES, a pesar de dar un rendimiento algo inferior, son más estables, y esto es debido a su comportamiento y su forma de balancear y repartir más la carga que el anterior.

### 4.4.3 – Escenario 3

El ultimo escenario que fue simulado es el que contiene un enlace LTE. De nuevo el módulo MPTCP reconoció ambos sub-flujos como se muestra en la figura 4.15.

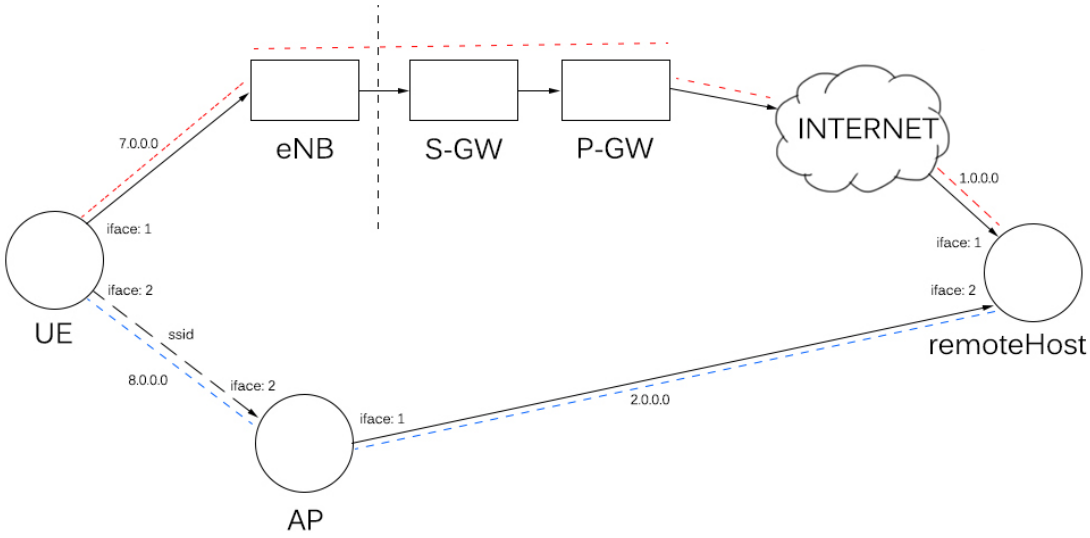


Figura *4.15 - Sub-flujos establecidos en el escenario 3*

De manera diferente al anterior, este caso presenta la peculiaridad de que los dos caminos disponibles poseen tecnologías diferentes, lo que implica que no es lo mismo que se encuentre congestionado el camino Wifi o el camino LTE de cara al rendimiento.

*Tabla 4.4 – Resultados del escenario 3 – caso 1*

| Caso 1 – Algoritmo UNCOUPLED | | |
|---|---|---|
| **Prob. de error en sub-flujo 1** | **Prob. de error en sub-flujo 2** | **Rendimiento** |
| 0.00001 | 0.00001 | 100 % |
| 0.00005 | 0.00001 | 72.3 % |
| 0.00005 | 0.00002 | 70.2 % |
| 0.0005 | 0.0009 | 9.3 % |

*Tabla 4.5 – Resultados del escenario 3 – caso 2*

| Caso 2 – Algoritmo FULLY COUPLED | | |
|---|---|---|
| **Prob. de error en sub-flujo 1** | **Prob. de error en sub-flujo 2** | **Rendimiento** |
| 0.00001 | 0.00001 | 77 % |
| 0.00005 | 0.00001 | 56.7 % |
| 0.00005 | 0.00002 | 54.8 % |
| 0.0005 | 0.0009 | 8.5 % |

*Tabla 4.6 – Resultados del escenario 3 – caso 3*

| Caso 3 – Algoritmo LINKED INCREASES | | |
|---|---|---|
| **Prob. de error en sub-flujo 1** | **Prob. de error en sub-flujo 2** | **Rendimiento** |
| 0.00001 | 0.00001 | 71.6 % |
| 0.00005 | 0.00001 | 55.6 % |
| 0.00005 | 0.00002 | 52.9 % |
| 0.0005 | 0.0009 | 9.8 % |

Poco queda por comentar de estos datos, pues el comportamiento que presenta es similar al visto en la sección 4.4.2.

# 5

# Conclusión

El último capítulo del documento abarca las conclusiones a las que se ha llegado tras analizar los resultados obtenidos en el capítulo anterior, así como todos los pensamientos que se han ido recopilando a lo largo del desarrollo del trabajo. En la última sección se dejarán abiertas líneas de investigación para posibles trabajos futuros que han ido surgiendo con el tiempo.

## 5.1 – Conclusiones obtenidas

Como ya se ha comentado, el mundo de las comunicaciones parece estar encaminándose por el modelo de comunicaciones multa-camino para poder aprovechar todos los recursos que se nos ofrecen hoy en día. Estas posibilidades multicamino abarcan, incluso, la posibilidad de emplear dos tecnologías de acceso diferente, como se ha visto en la sección 4.3.3, siendo esta una situación que no dista de la realidad, pues la mayoría de los usuarios la viven todos los días. Entre la multitud de soluciones que se le ha dado a este problema parece que es MPTCP la que más estabilidad presenta, respondiendo con buenos resultados en los estudios realizados.

Ya se han visto los estudios realizados de MPTCP en los que se estudia su rendimiento y su mejora frente al uso del TCP convencional, pero este trabajo quiere adentrarse en el estudio de MPTCP sobre redes inalámbricas y frente al escenario en el que dos redes de acceso de tecnologías diferentes se presentan ante el usuario. En estos escenarios, se quiere buscar la toma de decisión correcta sobre el camino a emplear frente a diferentes probabilidades. Es por eso que se han estudiado los diferentes algoritmos de control de congestión propuestos.

La respuesta que nos dan estos diferentes algoritmos es clara. Presentan la posibilidad de adaptarse a diferentes grados de congestión en la red garantizando una buena calidad de servicio al usuario. El protocolo MPTCP no ha respondió bien solo en estos casos, también nos

ofrece la posibilidad de balancear la carga de forma que se pueda aprovechar y exprimir al máximo las capacidades de la red, ventaja que puede ser muy atractiva de cara al proveedor de servicios. A esto se le añade la estabilidad y robustez que aporta el protocolo frente a situaciones indeseadas en la red o en un enlace determinado.

De los tres algoritmos estudiados, no se puede decidir cuál es mejor y cual es peor. Cada uno muestra prestaciones interesantes, características que pueden ser beneficiosas en determinadas situaciones. Si bien es cierto que el algoritmo UNCOUPLED parece peor que los otros, hay que tener en cuenta un factor que no se ha valorado en el trabajo y es la facilidad de su implementación.

Por ultimo hay que añadir que gran parte del desarrollo del trabajo se ha basado en la adaptación del módulo para ns-3 lo que permitirá futuras experimentaciones, así como la comprensión tanto de este módulo como del propio protocolo, que como ya se ha mencionado, parece que se verá en los próximos años.

## 5.2 – Líneas futuras

A lo largo del desarrollo de este trabajo han surgido ideas y cuestiones, que si bien, alguna se ha ido incorporando al proyecto otras han quedado en el tintero por diferentes motivos. Estas ideas dejan abiertas líneas que pueden ser tratadas o investigadas en trabajos futuros.

La primera idea que surge e la posibilidad de estudiar el comportamiento de MPTCP en interiores de edificios, haciendo uso del módulo que LTE presenta. Es decir, una situación más realista en la que el simulador modele el interior de un edificio en una ciudad y respecto a las diferentes posiciones de los puntos de acceso se pueda beneficiar más un camino que otro.

Otra idea que emerge es experimentar y cuantificar el rendimiento de MPTCP sobre una red Wifi en el uso de una aplicación de tiempo real como puede ser el *streaming* de un video.

En la sección 2.3 se estudió la propuesta de un algoritmo de *scheduling* óptimo. La implementación de este algoritmo sobre el módulo MPTPC del simulador ns-3 es interesante, estudiando de nuevo su rendimiento en topologías sencillas, cableadas o Wireless.

Otra idea más compleja de desarrollar seria la creación, e implementación de un nuevo algoritmo de control de congestión más un algoritmo de *scheduling* que dé respuesta a otras situaciones que no se han estudiado o buscando mejorar los resultados obtenidos hasta ahora. Sin duda este camino será explorado para más de un estudio en el futuro.

Por supuesto, la idea que conlleva el mayor esfuerzo, bajo un punto de vista personal, es la implementación de un módulo MPTCP completo para ns-3, que incluya todas las funcionalidades. Es una labor encomiable al tratarse de una tecnología que está dando sus primeros pasos y para la que cada instante surgen nuevos estudios. Una vez este maduro, más o menos, el protocolo MPTCP, la implementación del modelo para el simulador será una necesidad que deberá cubrirse tarde o temprano.

La ultima idea que se deja caer es la observación del protocolo en redes más complejas, topologías más elaboradas y simulaciones y escenarios con mayor número de detalles, que intenten reproducir situaciones concretas y en caso de que surja algún inconveniente en dicha

situación, tratar de buscar solución. Aunque existen numerosos estudios sobre este protocolo, se necesitan más resultados y aún quedan muchas posibilidades que explorar.

# Referencias


[1] Y. Dong, N. Pissinou and J. Wang. “Concurrency Handling in TCP”, May 2007.

[2] T. Hacker and B. Athey. “The End-to-End Performance Efects of Parallel TCP Sockets on a Lossy Wide-Area Network”, April 2002.

[3] Y. Hasegawa, I. Yamaguchi, T. Hama, H. Shimonishi and T. Murase. “Improved Data Distribution for Multipath TCP communication”, December 2005.

[4] L. Ong, J. Yoakum. “An Introduction to the Stream Control Transmission Protocol (SCTP)" IETF RFC 3286, May 2002.

[5] K. Rojviboonchai and H. Aida. “An evaluation of multi-path Transmission Control Protocol (MPTCP) with robust acknowledgement schemes”, October 2002.

[6] D. Sarkar. “A Concurrent Multipath TCP and Its Markov Model", June 2006.

[7] M. Zhang, J. Lai, A. Krishnamurthy, L. Peterson and R. Wang. “A transport layer approach for improving end-to-end performance and robustness using redundant paths”, June 2004.

[8] A. Ford, C. Raiciu and M. Handley. “TCP Extensions for Multipath Operation with Multiple Addresses", July 2010.

[9] C. Raiciu, D. Wischik and M. Handley, “Practical Congestion Control for Multipath Transport Protocols”, 2009.

[10] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky. “An Extension to the Selective Acknowledgement (SACK) Option for TCP,” IETF RFC 2883. July 2000.

[11] Ka-Cheong Leung, Victor O.K. LI and D. Yang. “An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions, and Challenges”, April 2007.

[12] R. Ludwig, M. Meyer, “The Eifel Detection Algorithm for TCP,” IETF RFC 3522, April 2003.

[13] P. Sarolahti, M. Kojo, K. Yamamoto, M. Hata. “Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP” IETF RFC 5682. September 2009.

[14] F. Yang, P. Amer, N.Ekiz "A Scheduler for Multipath TCP", IEEE, 2013

[15] Alcatel Lucent: White Paper “The LTE Network Architecture A comprehensive tutorial”

[16] M. Kheirkhak “MultiPath TCP (MPTCP) Implementation in ns-3”. https://github.com/mkheirkhah/mptcp

[17] ns-3 project “ns-3 Manual Release ns-3.18”, August 2013