ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Proyecto Fin de Carrera

Combinación de técnicas de Network Coding y codificación lineal aleatoria sobre redes inalámbricas malladas

(On the combination of Network Coding techniques and random linear coding over wireless mesh networks)

Para acceder al Titulo de

INGENIERO DE TELECOMUNICACIÓN

Autor: Eduardo Rodríguez Maza

Junio - 2014

E.T.S. DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACION

INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

Realizado por: Eduardo Rodríguez Maza
Director del PFC: Ramón Agüero Calvo, David Gómez Fernández
Título: "Combinación de técnicas de Network Coding y codificación lineal aleatoria sobre redes malladas inalámbricas"
Title: "On the combination of Network Coding techniques and random linear coding over wireless mesh networks"

Presentado a examen el día: 6 de Junio de 2014

para acceder al Título de

INGENIERO DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Muñoz Gutiérrez, Luis
Secretario (Apellidos, Nombre): Agüero Calvo, Ramón
Vocal (Apellidos, Nombre): Beivide Palacio, Julio Ramón

Este Tribunal ha resuelto otorgar la calificación de: .....

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

Fdo.: El Director del PFC
(sólo si es distinto del Secretario)

Vº Bº del Subdirector

Proyecto Fin de Carrera Nº
(a asignar por Secretaría)

Índice

1. Introducción **1**

1.1. Planteamiento del problema 1

1.2. Objetivos del proyecto 2

1.3. Estructura de la memoria 3

2. Estado del Arte **4**

2.1. Redes Malladas 4

2.2. Network Coding 6

2.3. Inter-flow Network Coding 7

2.4. Intra-flow Network Coding 10

2.4.1. Random Linear Coding 11

2.4.2. Recodificación en nodos intermedios 16

2.4.3. Overhearing, enrutamiento oportunista 17

2.5. Antecedentes 18

2.6. User Datagram Protocol (UDP) 19

3. Teoría e Implementación **22**

3.1. Análisis del rendimiento 22

3.1.1. Rendimiento bruto de RLC 22

3.1.2. Rendimiento sobre enlaces IEEE 802.11b 24

3.1.3. Recodificación de paquetes 25

3.2. Canal 26

4. Network Simulator **27**

4.1. Características 27
4.2. Componentes 29
4.2.1. Aplicación 30
4.2.2. Transporte 30
4.2.3. Red 31
4.2.4. Enlace y físico 33
4.3. Desarrollo en el marco ns-3 34
4.4. Problemática 46
4.5. Soluciones adoptadas 46

5. Simulaciones y Resultados **49**
5.1. Conceptos 49
5.2. Proceso de medida 51
5.3. Escenarios 52
5.3.1. **Escenario de dos nodos** 52
5.3.2. **Escenario de tres nodos** 52
5.3.3. **Escenario multi-nodo** 53
5.4. Resultados 54
5.4.1. **Escenario de dos nodos** 54
5.4.2. **Escenario de tres nodos** 63
5.4.3. **Escenario multi-nodo** 66

6. Conclusiones y Líneas Futuras **74**
6.1. Conclusiones 74
6.2. Líneas futuras 77

Acrónimos **81**

Índice de Figuras

2.1. Red mallada inalámbrica 5

2.2. Escenarios de *Inter-Flow Network Coding* 8

2.3. Conexión de *Network Coding* con otras disciplinas 9

2.4. *RLC* en escenario de dos nodos 12

2.5. Eventos en recepción ($K = 8$) 13

2.6. Esquema de cambio de arquitectura 14

2.7. Formato de cabecera RLC 14

2.8. *RLC* en escenario de tres nodos 16

2.9. Escenario canónico de intra-flow *Network Coding* (NC) 17

2.10. Escenario 4 nodos de intra-flow NC 18

2.11. Cabecera UDP 20

2.12. Datos para el cálculo del checksum 21

4.1. Jerarquía de organización de ns-3 29

4.2. Esquemas de recorrido de los paquetes en ns-3 32

4.3. Wi-Fi en ns-3 33

4.4. Disposición espacial de los nodos 35

4.5. Configuración de los enlaces del despliegue 36

4.6. Secuencia de llamadas a funciones en una transmisión 44

4.7. Flujo de los paquetes a través de la pila de protocolos de ns-3 48

5.1. Escenario de dos nodos 52

5.2. Escenarios de tres nodos 53

5.3. Escenario multi-nodo 54

5.4. Throughput para cada valor de K y Q 55
5.5. Rendimiento máximo en el canal 56
5.6. Throughput en un canal sin pérdidas 56
5.7. Ratio de paquetes en exceso 57
5.8. Penalización por paquetes en exceso 58
5.9. Penalización por el envío de ACK's 58
5.10. Factores de penalización 59
5.11. Retardo entre fragmentos consecutivos 60
5.12. Tiempo de cálculo del rango 61
5.13. Tiempo de inversión de la matriz 61
5.14. Influencia de las retransmisiones 802.11 en un canal con pérdidas 63
5.15. Impacto del overhearing en store-and-forward (RLSC) 64
5.16. Impacto del overhearing en network coding (RLNC) 64
5.17. Throughput en un canal con pérdidas 66
5.18. Comparación de cuatro escenarios de retransmisión multinodo 67
5.19. Escenarios con *overhearing* deshabilitado 69
5.20. Escenarios con *overhearing* habilitado 70
5.21. cdf's de los escenarios con 4 nodos intermedios 71

Índice de Tablas

4.1. Fichero de configuración espacial de los nodos 35
4.2. Fichero de configuración del canal 35
4.3. Fichero de configuración de rutas 36
4.4. Formato de la primera versión del fichero de traza resumida 38
4.5. Formato de la segunda versión del fichero de traza resumida 38
4.6. Formato de traza temporal 39

Resumen ejecutivo

En este documento se describe el estudio que se ha desarrollado para analizar la influencia de las técnicas de network coding, combinadas con codificación lineal aleatoria, en el rendimiento en servicios de transmisión fiables sobre redes malladas inalámbricas. Concretamente, se aplicarán sobre un único flujo de datos (*intra-flow*) y utilizando UDP como protocolo de transporte. Este tipo de técnicas se basan en que los nodos intermedios dejan de tener un papel pasivo, fomentando una mejora del rendimiento e incrementando la robustez de la transmisión.

El fuerte crecimiento en el uso de tecnologías inalámbricas ha propiciado constantes avances en este campo. Entre ellos destacan las redes malladas inalámbricas, que se han consolidado como una alternativa barata y sencilla para extender la cobertura de despliegues tradicionales. Sus características hacen que puedan beneficiarse de las ventajas ofrecidas por *network coding*. Sin embargo, este tipo de técnicas no son aún maduras y no existen numerosos estudios sobre las posibilidades reales de su uso.

Para poder conocer los beneficios y problemas que supone el uso de técnicas de *network coding*, se ha hecho uso de la herramienta *ns-3*. Tras una extensa campaña de simulaciones, se han observado mejoras, en términos de throughput, de hasta un 200 % en comparación con TCP.

Palabras clave: Network Coding, Random Linear Coding, UDP, Network Simulator 3, Wireless Mesh Network

Abstract

This report presents a study of the benefits brought about by the use of network coding techniques, together with random linear coding, regarding the performance of reliable communication services over wireless mesh networks. In particular, we work with a single data flow (intra-flow) and we use UDP as the transport layer protocol. This kind of techniques foster the intermediate nodes to take a more active role, helping to improve the achieve performance and the reliability of the communications.

The remarkable growth of wireless technologies usage has contributed to continuous progresses in this field. An outstanding example is the so-called wireless mesh networks, which have become a cheap and simple alternative to extend the coverage of traditional network deployments. Their characteristics make them appropriate to assess the advantages offered by network coding techniques. However, this type of solutions are not mature enough and there do not exist many studies on the real possibilities of their performance.

In order to understand the benefits and problems related to network coding, we have used the ns-3 simulator. By means of a thorough simulation study, we have assessed throughput improvements up to 200 %, as compared with the legacy TCP.

Keywords: Network Coding, Random Linear Coding, UDP, Network Simulator 3, Wireless Mesh Network

Agradecimientos

En primer lugar, quiero agradecer a Ramón Agüero la oportunidad de realizar este proyecto. Su gran labor académica y su dedicación han sido, en gran parte, lo que me ha conducido aquí.

Tampoco quiero olvidarme de David Gómez, sin cuya ayuda esto no habría sido posible. Muchas gracias por ayudarme y por tratar de resolver cada uno de mis problemas y dudas por banales que fueran. Ambos sois un referente de lo que me gustaría llegar a ser.

Al resto de compañeros del laboratorio, por el estupendo ambiente de trabajo en el que me han integrado y por hacer más llevaderos los días más duros.

A mis compañeros de la carrera, que han hecho de estos años un cúmulo de buenos recuerdos y anécdotas. Porque más que compañeros, son amigos.

A mis amigos de siempre, por todos los buenos momentos y porque cada vez que nos vemos parece que no ha pasado el tiempo.

A mi familia, en especial a mis padres, que han hecho todo lo posible para que pueda estar donde estoy. Sin ellos no sería la persona que soy. Muchísimas gracias por todo.

A Adriana, por apoyar cada una de mis decisiones y ser el punto de apoyo que toda persona necesita. Gracias por regalarme una sonrisa cada día.

Por último, a todo el que lea este proyecto, espero que lo disfrute, al menos, una mínima parte de lo que lo he disfrutado yo realizándolo.

Gracias.

1 Introducción

En este primer capítulo se plantean los motivos que han conducido a la propuesta y realización de este proyecto de fin de carrera. En base a esto, se definen sus objetivos. Por último, en una tercera sección, se explicará como está estructurada la memoria del proyecto, mostrando una breve explicación de cada uno de los capítulos en los que se ha dividido.

1.1 Planteamiento del problema

Desde los inicios del Siglo XXI se ha producido un continuo crecimiento de las tecnologías inalámbricas aplicadas a las comunicaciones en múltiples aspectos: volumen de ventas, usuarios, fabricantes, estandarización, etc. Este hecho, junto a las ventajas de movilidad, disminución del cableado y comodidad para el usuario, han provocado que la tecnología esté en continua evolución y desarrollo, convirtiéndola en la alternativa de acceso a comunicaciones más extendida en la actualidad.

Una de las aplicaciones con mayor proyección de futuro son las redes malladas inalámbricas (*Wireless Mesh Networks* (WMN)). Este tipo de topología, de la que se hablará más adelante, se ha convertido en una solución ampliamente adoptada por los operadores como método para ampliar su cobertura de una manera económica. Estas redes proveen de un entorno multi-camino y multi-salto, que proporciona capacidad de recuperación ante fallos puntuales en nodos de la red, en contraste con otras alternativas más centralizadas. Ante este escenario surge la necesidad de afrontar nuevos retos que repercutirán en la evolución de esta tecnología.

A pesar de sus evidentes ventajas, la tecnología inalámbrica no está exenta de ciertos problemas, que pueden poner en peligro el éxito de su despliegue. Las comunicaciones por aire son susceptibles de ser interceptadas, sufrir interferencias por el uso compartido de bandas de frecuencia y, además, acusan el hecho de no poder alcanzar velocidades de

transmisión equiparables a un medio cableado. Además, comportamiento del protocolo *Transmission Control Protocol* (TCP) sobre redes inalámbricas se ve perjudicado, debido a que en su origen fue concebido para trabajar sobre redes más clásicas.

Con el surgimiento en la última década de técnicas de NC, concebidas para paliar estos problemas, uno de los retos inmediatos a tener en cuenta es el uso y optimización de ellas. Las pretensiones de estos métodos son proporcionar una solución alternativa para las comunicaciones extremo a extremo a nivel de transporte.

1.2 Objetivos del proyecto

El objetivo principal de este trabajo es el de realizar un estudio detallado de como influye el uso de técnicas de NC sobre un protocolo de transporte no orientado a conexión, en este caso *User Datagram Protocol* (UDP), en redes inalámbricas multi-salto. El proyecto se centrará principalmente en NC aplicado a comunicaciones en las que la codificación conjunta de paquetes se realiza sólo entre elementos de un mismo flujo de datos, *intra-flow* NC.

Se analizará el impacto de los distintos parámetros del protocolo, como puede ser el tamaño de fragmento, el orden del campo de Galois utilizado para la codificación de los paquetes, el uso de codificación fuente o la codificación en los nodos intermedios. El estudio se realizará sobre topologías relativamente sencillas, pero significativas, cuya finalidad es comprobar la viabilidad de la solución propuesta en escenarios de mayor complejidad, estudiando también como influyen los parámetros inherentes al canal, y el escenario en el rendimiento global de la transmisión.

El entorno de trabajo a utilizar es *Network Simulator 3* (ns-3). Este simulador no dispone de un módulo concreto para implementar NC, por lo que se hace necesario el desarrollo e inclusión de una serie de entidades particulares, aprovechando las características del simulador, para comprobar el comportamiento del protocolo y describir las interacciones de esta capa con el resto de elementos presentes en la comunicación.

El proyecto se enmarca en un línea de investigación activa en el Grupo de Ingeniería Telemática (GIT), que la está explotando en el marco del proyecto de investigación **COSAIF**, *"Connectivity as a Service: Access for the Internet of the Future"* (TEC2012-38754-C02-02).

1.3 Estructura de la memoria

A continuación se explica la estructura del documento, presentando brevemente el contenido de cada uno de los capítulos que lo componen:

- En el Capítulo 2 se expondrán distintos trabajos realizados previamente en el campo de NC,bosquejando la situación actual de la investigación en estos momentos. Además, se describen los diversos conceptos teóricos que sostienen la línea de investigación de este proyecto, y se explican elementos fundamentales para la comprensión de la mecánica de funcionamiento de NC tales como: sus diferentes variedades, la recodificación en nodos intermedios y el *overhearing* de paquetes. El objetivo de este capítulo es proporcionar las bases necesarias para asimilar adecuadamente el desarrollo realizado a lo largo del proyecto.
- El Capítulo 3 sirve para introducir una serie de desarrollos teóricos necesarios para la caracterización del comportamiento de las técnicas de NC en los diferentes escenarios estudiados en el proyecto. Para posteriormente compararlo con los resultados obtenidos en las simulaciones. También se explica el comportamiento del canal utilizado en la campaña de pruebas realizada.
- En el Capítulo 4 se realiza un profundo análisis de la principal herramienta utilizada en este proyecto, el simulador de redes ns-3. Se explica su estructura modular y se desgranan los principales componentes involucrados en el proceso de comunicación. Tras ello, se exponen los cambios y el desarrollo que ha sido necesario incluir tanto del protocolo NC como de sus funcionalidades, en el entorno del simulador. Por último, se aborda la problemática surgida a la hora de desarrollar y hacer funcionar los nuevos módulos en el simulador, y cómo finalmente se han logrado implementar las soluciones para subsanar estos problemas.
- El Capítulo 5 es posiblemente la parte central del proyecto. Inicialmente se realiza un repaso de una serie de conceptos básicos para estudiar de una manera correcta los resultados obtenidos en el simulador. Además, se explica detalladamente la campaña de medidas que se ha llevado a cabo para realizar la correcta caracterización del protocolo. Se muestran y analizan los escenarios en los que se han efectuado las pruebas. Por último, se describen los resultados obtenidos con la simulación, y se analiza la validez y versosimilitud con el análisis teórico previo. También se hace una reflexión sobre la viabilidad de este método frente al previo TCP en términos de rendimiento, recursos etc.
- Por último, en el Capítulo 6, se detallan las conclusiones alcanzadas con el proyecto tras el análisis de las simulaciones. Se realiza un resumen de los principales aspectos del protocolo y se exponen una serie de líneas futuras de interés que quedan abiertas tras la finalización de este trabajo.

2

Estado del Arte

Este capítulo recoge la introducción de cada uno de los elementos principales que componen el trabajo. De esta forma el lector puede adquirir unos conocimientos básicos que le permitan abordar y comprender correctamente las distintas cuestiones tratadas a lo largo de las siguientes páginas.

2.1 Redes Malladas

Las redes malladas inalámbricas (*Wireless Mesh Networks*, WMN) son una alternativa de comunicación reciente cuya utilidad en ciertos entornos o condiciones puede ser muy grande. Por ejemplo, una zona catastrófica, como Haití tras el Huracán Katrina, o zonas donde, deliberadamente por razones políticas, se limita la comunicación a través de redes como Internet, Egipto o Siria durante los recientes conflictos, son algunos de los ámbitos donde estas redes podrían ser de gran utilidad. La posibilidad de desplegar, rápidamente y mediante los dispositivos disponibles (ordenadores personales, teléfonos móviles etc.), una red independiente de terceros proveedores con la que comunicarse para ayudar en las labores de rescate e información es una herramienta muy poderosa y que, sin duda, ayudaría en gran medida en las labores pertinentes en estos escenarios.

Las redes más tradicionales se basan en un entorno centralizado en el que el tráfico se dirige a puntos de acceso o *gateways*, desde donde es redirigido al destino. Por el contrario, una WMN se comporta de una manera similar a una red *ad-hoc*, en la que cada nodo de la red no actúa únicamente como origen o destino, sino que toman un papel más activo, al poder comportarse como routers (*mesh routers*) dentro de la red. Una de las posibilidades que abren es que se pueda ofrecer servicio en zonas aisladas a las que antes no era posible, a través del enrutamiento del tráfico a través de otros nodos (cuando sus zonas de cobertura se solapan) hasta llegar a un *gateway* o punto de acceso.

Las principales ventajas que presentan son su escalabilidad y fiabilidad. Añadir nuevos

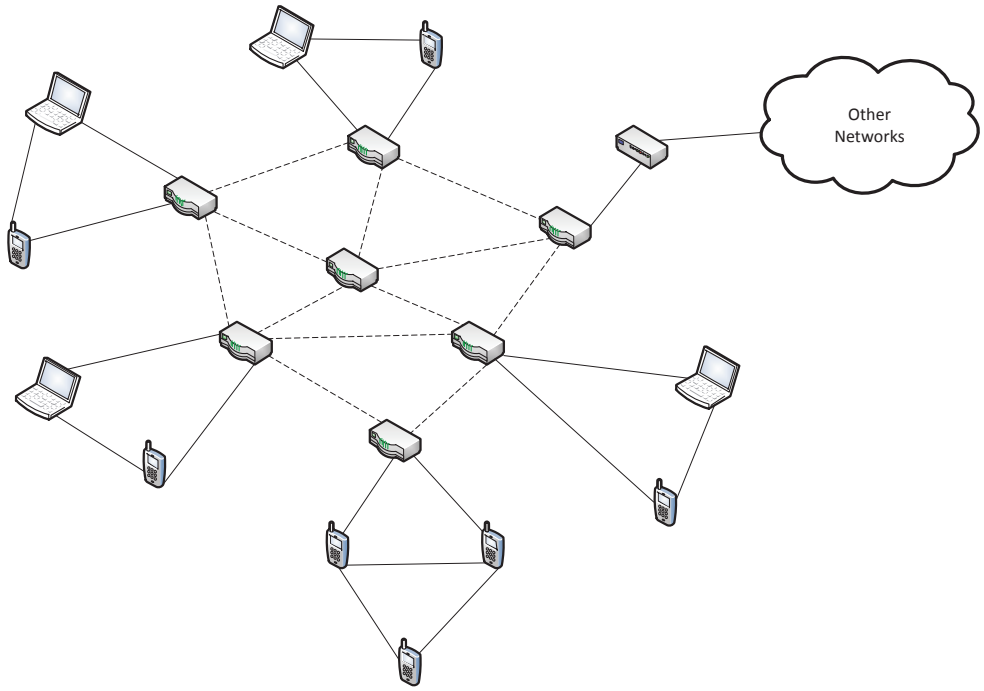


Figura 2.1: Red mallada inalámbrica

elementos a la red conforme aumentan las necesidades de la misma (mayor número de usuarios, expansión geográfica, aumento de capacidades) se convierte en una preocupación menor, en comparación con el proceso necesario en redes tradicionales, en las que añadir nuevos elementos al escenario puede requerir de un redimensionamiento o rediseño de la red. Las WMN son redes autoconfigurables, que se adaptan a cada cambio que ocurre en ellas, por lo que el hecho de incorporar un nuevo nodo a la red es un proceso automático. Por otro lado, cabe destacar la robustez de estas topologías. Las WMN se comportan de una manera fiable ante fallos puntuales en nodos, ya que el tráfico puede llegar a su destino por múltiples rutas. Por último, puesto que los nodos de la red pueden ejercer labores de *routing*, es posible aprovechar oportunidades de *overhearing*, y con ello mejorar el *throughput* de la transmisión de información.

Un elemento a destacar es el aspecto social que tienen estas redes. Representan una forma barata y eficiente para conectarse y comunicarse con una comunidad más amplia. Cada miembro de la red contribuye con sus recursos, no habiendo un "dueño" ni administrador de la red. También contribuyen al anonimato de la red, ya que es muy complicado discernir la identidad real de los usuarios conectados a ella; además, para monitorizar el tráfico, hay que estar conectado a la propia red.

Sin embargo, también existen una serie de desventajas. En primer lugar, a medida que la red crece, el número de saltos necesarios para que la información de un nodo alcance su destino (otro nodo, un *gateway* para salir hacia el exterior, etc.) también aumenta. Por tanto, el throughput se ve reducido de una manera relevante cuando la ruta crece

en exceso (en el Capítulo 5 se podrá observar este efecto). El *overhearing* de paquetes puede ayudar a mejorar la capacidad en estos casos, como se verá en la Sección 2.4.3. La otra gran desventaja es la penalización sobre el rendimiento que producen los entornos inalámbricos en combinación con protocolos que originalmente estaban planteados para medios de transmisión fiables, con baja tasa de fallos, como es TCP. La utilización de técnicas de NC combinadas con UDP como protocolo no orientado a conexión permite solventar esta deficiencia, por lo que el proyecto se centra en este método.

2.2 Network Coding

El NC, como campo de estudio, es relativamente joven. Es en el año 2000 cuando, en la publicación de Ahlswede, Cai, Li, and Yeung [1], se considera el nacimiento de NC. Como otras muchas nuevas materias de estudio, se caracteriza por un fuerte debate inicial en cuanto a las posibilidades de este método, y al escepticismo en torno a su aplicación real.

Se espera que en el futuro, tras constatar su validez, NC se convierta en una técnica ampliamente extendida en WMN. A lo largo de esta sección se tratará de arrojar un poco de luz acerca de esta técnica, sus ventajas, sus problemas y sus posibles aplicaciones.

Definir NC de una manera directa no es sencillo. En [1], Ahlswede, Cai, Li, and Yeung se refieren a NC como *"codificar en un nodo de la red"*, entendiendo *"codificar"* como un mapeo causal arbitrario de las entradas a las salidas. A medida que se ha refinado la técnica han surgido definiciones algo más concretas como *"codificar en un nodo de una red de paquetes"* donde la información a transmitir es dividida en paquetes sobre cuyo contenido se aplica NC. Esta última definición es más cercana a la realidad que la inicialmente dada por Ahlswede, Cai, Li, and Yeung. A continuación se procede a detallar esta aproximación.

Antes de la proposición de este tipo de técnica, los nodos de la red no realizaban ningún tratamiento de la información que recibían, y se limitaban a retransmitir los datos haciendo las modificaciones necesarias en las cabeceras de los paquetes. Sin embargo, el aumento de la capacidad de cálculo y almacenamiento en los dispositivos, junto con las mejoras en técnicas de codificación, han permitido que los nodos intermedios no sean meros testigos del proceso de transmisión de información entre el origen y el destino, y puedan tomar un papel activo en el proceso, con la consecuente mejora de rendimiento, como se verá en el Capítulo 5.

NC se podría definir como una capa intermedia entre la de transporte y la de red. Básicamente, su función es la de combinar los paquetes a transmitir para generar nueva información, que no existe explícitamente en los paquetes originales y, así, disminuir el número de transmisiones. Este proceso se puede realizar de varias formas, que se estudiarán con más detalle en las Secciones 2.3 y 2.4. La combinación de los paquetes requiere de

técnicas de codificación tanto en el origen como en los nodos intermedios, procesamiento en los nodos intermedios y decodificación en el destino. El uso de NC abre un nuevo rango de posibilidades, más allá de la mejora de rendimiento: la reducción de transmisiones contribuye al ahorro energético en el dispositivo, con lo que la batería tiene una duración mayor, puede proporcionar un servicio más fiable, e introduce nuevos mecanismos de seguridad (en algún modo la información está encriptada).

En la siguiente sección se distinguirá entre los distintos tipos de NC que existen, las situaciones en las que se aplica cada uno y el esquema de codificación que utilizan.

Una de las claves de NC es que aprovecha la característica *broadcast* del medio inalámbrico. Hasta ahora, las comunicaciones *wireless* seguían enfocadas a transmitir de un nodo a otro de manera *unicast*, como en un escenario cableado. Esta mejora permite realizar una "escucha oportunista" de paquetes. Se asume que los nodos vecinos al destinatario del paquete, en un salto concreto pueden recibir y almacenar el paquete. Gracias a ello, nodos que inicialmente no eran los destinatarios del paquete pueden hacer uso de la información para codificar.

En función de que información se combine y cómo se codifique se hablará de dos tipos de NC: *inter-flow* NC e *intra-flow* NC.

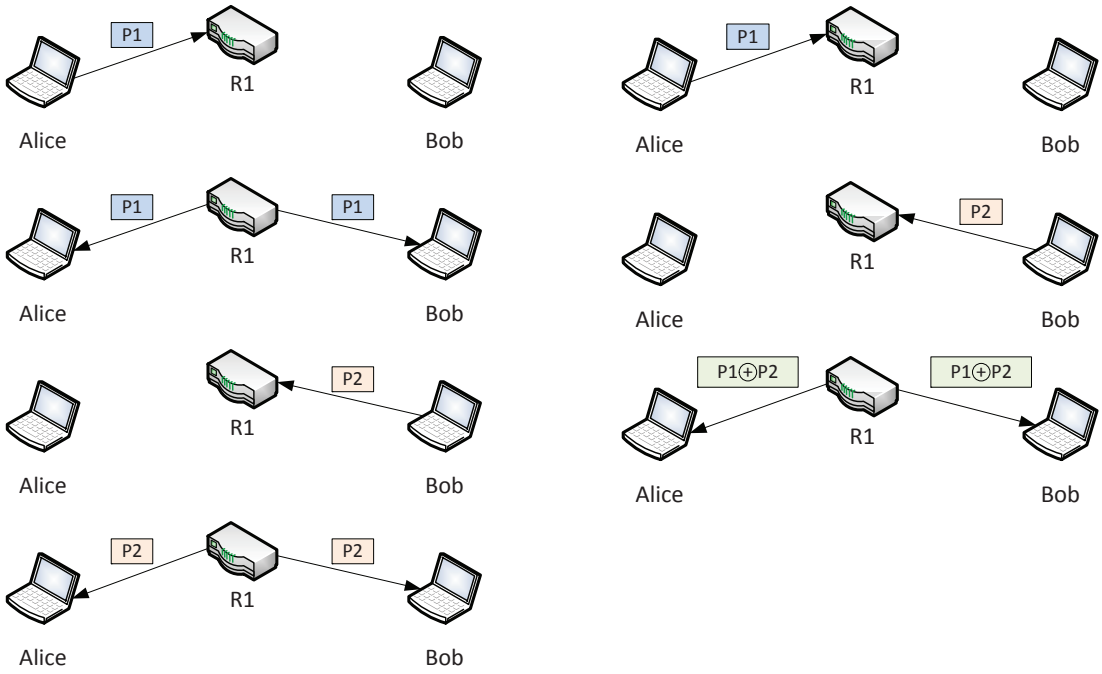
2.3 Inter-flow Network Coding

En este apartado se proporciona una breve explicación del inter-flow NC. Si bien, no es el objetivo principal de este proyecto, conviene tratar el tema para tener una comprensión más global de los esquemas de NC.

En este método, un nodo realizando "escucha oportunista", a partir de ahora *coding node*, recibe paquetes de varios flujos de datos independientes, los procesa de manera conjunta y los transmite a los destinatarios originales en modo *pseudo-broadcast*. Los destinos de esos paquetes reciben los datos codificados y tratan de realizar el proceso de decodificación mediante la información de la que ya disponen. Se pretende que el *coding node* ahorre un cierto número de transmisiones.

Debido a los retardos aleatorios en la red, las oportunidades para codificar varios paquetes conjuntamente es bastante baja. Por eso, se introduce el parámetro *Coding Time* (C_T), que funciona como un temporizador que determina el tiempo máximo que se puede tener un paquete almacenado en el *coding node* sin combinarlo con otros paquetes. Una vez superado este tiempo el paquete se transmite sin codificación alguna. Gracias a esto se evitan congestiones innecesarias por una espera demasiado larga y por un almacenamiento innecesario en el buffer.

El otro parámetro importante de este tipo de codificación es el *Buffer Size* (B_S). Su cometido es determinar la cantidad de paquetes que se pueden almacenar en el *coding node* a la espera de una oportunidad de codificación. Al igual que con el C_T, se busca aumentar las posibilidades de codificar paquetes y, a la vez, no congestionar en exceso los nodos intermedios.



(a) Escenario sin NC

(b) Escenario con NC

Figura 2.2: Escenarios de *Inter-Flow Network Coding*

En cuanto a la codificación que realizan los *coding nodes* existen múltiples opciones. Aquí se tratará un caso sencillo en el que la operación que se aplica sobre los paquetes es una *Exclusive OR* (XOR). Así, cuando el destino trate de recuperar la información que le interesa no tendrá más que aplicar esa misma operación sobre los paquetes nativos que ya tiene almacenados, y el paquete codificado que ha recibido. Esto tiene una ventaja añadida y es que refuerza la seguridad de la transmisión inalámbrica, ya que es necesario conocer los paquetes nativos, el tipo de codificación que se utiliza y, además, el paquete codificado, para poder interceptar una información.

En la Figura 2.2 se muestra un ejemplo sencillo que ayudará a comprender el proceso con mayor claridad. Se presenta un escenario (Figura 2.2a) en el que existen dos equipos (Alice y Bob) que se disponen a transmitir información entre ellos. Existe además un nodo intermedio que reenvía los paquetes nativos que ambos transmiten.

En primer lugar, ocupando una ranura temporal de transmisión, Alice transmite su paquete al nodo intermedio. Posteriormente este lo reenvía hacia Bob. Tras ello, en un tercer slot temporal, Bob se dispone a entregar el paquete al router. Finalmente, igual que

lo ocurrido con Alice, el nodo intermedio entrega el paquete a su destino (Alice). En total el envío de ambos paquetes ha requerido cuatro transmisiones.

La Figura 2.2b representa el mismo proceso de comunicación, utilizando NC en el nodo intermedio. En el instante inicial, Alice envía el paquete nativo al router. El paquete se retiene en el nodo a la espera de una oportunidad de codificación. En el siguiente slot temporal Bob transmite su paquete, que se recibe en el *coding node*. En este momento surge una oportunidad de codificación con los paquetes *P1* y *P2* y, por tanto, *R1* procede a realizar la operación XOR sobre ellos transmitiéndolos al canal en modo *broadcast*. Debido a que la transmisión es inalámbrica, ambos equipos reciben el paquete codificado al mismo tiempo, y son capaces de recuperar la información original, ya que poseen el paquete nativo que ellos mismos habían transmitido. En este caso el número de transmisiones necesarias para completar el envío de la información ha sido tres.

A la vista de esto se pueden observar algunas de las bondades de esta técnica. En primer lugar, la reducción de transmisiones, que conlleva un ahorro de energía en el nodo intermedio. También se aprecia una mejora del *throughput*, ya que se transmite el mismo volumen de información, pero el tiempo total para completar el envío se ha reducido de cuatro slots temporales a tres. Por último, el retardo asociado a los paquetes se reduce, debido a que la retransmisión de los paquetes no es inmediata (espera a una oportunidad de codificación), dejando libre el canal para el envío de paquetes provenientes de otras fuentes.

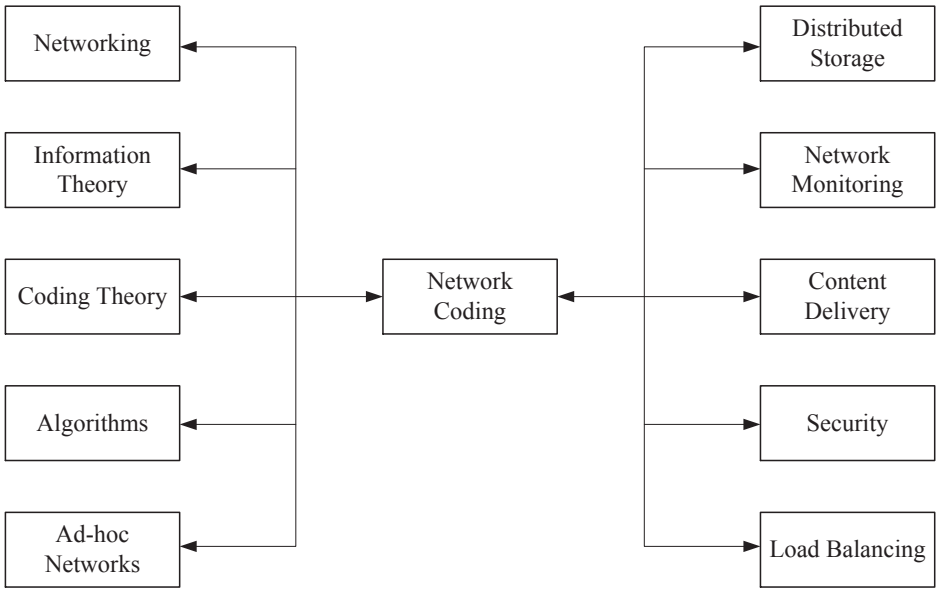


Figura 2.3: Conexión de *Network Coding* con otras disciplinas

Por otra parte, este modalidad de NC provoca interés en otras disciplinas. La Figura 2.3 muestra un esquema que relaciona los campos de estudio de la técnica, así como los

ámbitos donde es aplicable. La monitorización de redes, el almacenamiento distribuido o la seguridad son algunos de ellos.

Actualmente, en entornos de redes, es necesario que los nodos mantengan un informe del estado de los enlaces, con el fin de utilizar los recursos de manera eficiente y evitar errores. Para conocer las características de la red, topología, retardo, número de saltos, se utilizan medidas extremo a extremo, por lo que el intercambio de paquetes de sondeo debería ser lo más ligero posible, con el fin de no sobrecargar en exceso la red. En este ámbito, se ha estudiado que el uso de NC permite, con el mismo número de transmisiones, determinar la topología de una red con más saltos. Por tanto, el ancho de banda disponible para transmitir información es mayor, y la sobrecarga que provocan estas tareas se reduce.

En cuanto al tema de la seguridad, el hecho de codificar los paquetes (combinaciones lineales de los nativos) confiere una mayor robustez a la información ante posibles escuchas, mientras trata de garantizar, mediante la transmisión multi-camino y la distribución en modo *broadcast*, la entrega de la información en el destino.

2.4 Intra-flow Network Coding

En este apartado se tratará la técnica de NC en la que se ha centrado este proyecto, el *intra-flow* NC. Se procederá a explicar los mecanismos que intervienen en el proceso de transmisión de datos con este método. Más adelante, en el Capítulo 3, se estudiarán algunos aspectos teóricos adicionales relativos a esta técnica.

Dentro de la disciplina de NC se puede realizar un tratamiento de la información perteneciente a distintos flujos de datos (*inter-flow* NC) o al mismo flujo de datos (*intra-flow* NC) como es este caso. La aplicabilidad de este método concreto está ligada a las redes de paquetes. En ellas, la información a transmitir se trocea en fragmentos de longitud fija, para luego mandarlos hasta su destino. En función de la topología de la red y de como esté configurada, la forma en que los paquetes alcanzan su destino varía. La principal ventaja es que, ante posibles fallos a lo largo de la ruta, la información que se pierde es solamente un fragmento del total, siendo relativamente fácil recuperarla si fuera necesario. Sin embargo, esto requiere de una serie de funcionalidades para su correcto funcionamiento como: control de flujo, control de errores, ordenación de paquetes etc.

Actualmente el protocolo de transporte más extendido es TCP. Sin embargo, por su concepción original, su rendimiento en redes inalámbricas es deficiente. Cuando se propuso TCP (en los años 70) prácticamente la totalidad de las redes estaban basadas en entornos cableados y, consecuentemente, TCP se ajusta a las necesidades de estas redes. El mecanismo de retransmisión de paquetes erróneos, que es una incidencia habitual en redes inalámbricas, penaliza en exceso el rendimiento, debido a la lenta reacción de TCP a través de la técnica de *congestion avoidance*. Además, el sistema de secuenciamiento de

paquetes puede eliminar mucha información, ya que la pérdida de un segmento provoca que se deseche mucha información que llegue en paquetes posteriores.

Estas desventajas son bien conocidas por la comunidad investigadora, que ha hecho un gran esfuerzo por proponer alternativas para subsanar los problemas mencionados con anterioridad. Se han estudiado multitud de posibilidades: realizar modificaciones en TCP para adecuarlo a las exigencias de un medio inalámbrico, diseñar nuevos protocolos de transporte plenamente capacitados para lidiar con los problemas mencionados (por ejemplo *Stream Control Transmission Protocol* (SCTP) [2]) o soluciones mediante técnicas *cross-layer* como NC.

2.4.1. Random Linear Coding

La solución que se adopta en este proyecto es una combinación de UDP con *Random Linear Coding* (RLC), combina datagramas que pertenecen al mismo flujo de datos para lograr una transmisión fiable y eficiente de la información. Posteriormente esta idea inicial evolucionará al ya mencionado *intra-flow* NC. A continuación, se procede a describir las características de esta técnica.

Para ilustrar la explicación, se utilizará el ejemplo sencillo que se muestra en la Figura 2.4. En este escenario, el nodo origen dispone de un fichero de datos a transmitir hacia el destino. En una situación normal el fichero se trocearía en paquetes de igual longitud y se enviarían secuencialmente. Una vez el destino tuviera todos, recompondría el fichero original. Por el contrario, el esquema propuesto toma un número determinado (K) de los paquetes nativos (a cada uno de estos conjuntos se les denominará, a partir de ahora, fragmentos) los almacena en un buffer de transmisión, multiplica cada uno por un factor aleatorio y los combina en un único paquete de la misma longitud que cada uno de los nativos. Por tanto, un paquete codificado será de la forma $p' = \sum_{i=0}^{K-1} c_i \cdot p_i$, donde los c_i son coeficientes aleatorios generados desde un campo finito de Galois $GF(Q) = GF(2^q)$ y los p_i son los paquetes nativos. Estos coeficientes aleatorios pueden representarse también como: $\overrightarrow{c} = (c_0, c_1, \cdots, c_{K-1})$, siendo $\overrightarrow{c}$ el vector de codificación del paquete. La capa de NC enviará de manera periódica paquetes codificados hasta que el destino confirme la correcta recepción y decodificación del fragmento completo.

En el otro extremo de la comunicación, el nodo destino hace uso de dos entidades para almacenar la información necesaria. La primera de ellas es una matriz C, de dimensión $K \times K$, que almacena los vectores de codificación (introducidos por filas) de los paquetes. La segunda es un buffer de recepción, capaz de guardar K paquetes a la espera de poder decodificarlos.

Una vez que se recibe un paquete codificado p'_i en el destino, su vector codificado se introducirá en la fila j-ésima de la matriz, y comprobando si aporta nueva información a

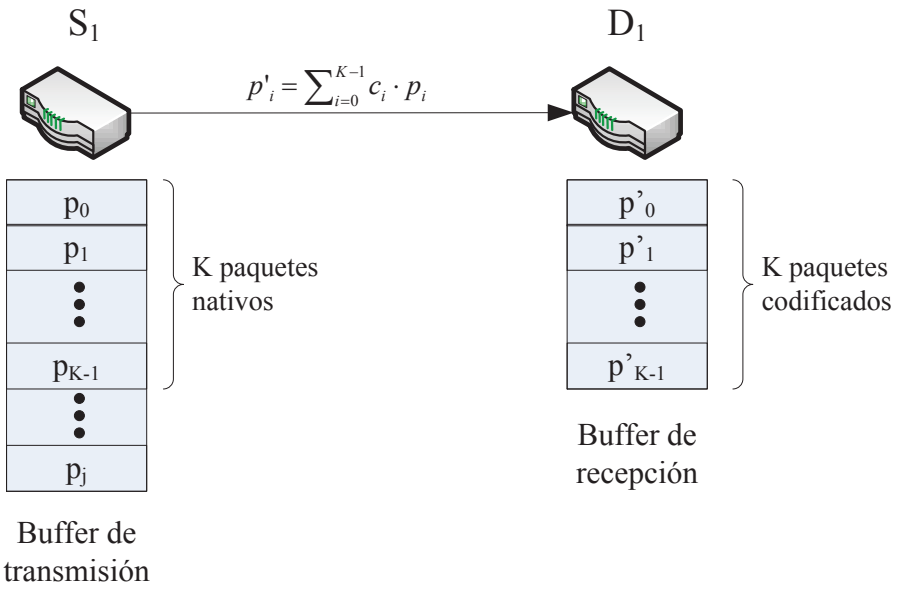


Figura 2.4: *RLC* en escenario de dos nodos

la que ya tiene el nodo destino mediante el cálculo del rango de la matriz. En el caso de que la información sea novedosa, el paquete se almacena en el buffer de recepción; en caso contrario, el vector será eliminado de la matriz C. Se puede observar que cada paquete codificado puede contener información de cualquiera de los K utilizados para conformarlo. Por tanto, cada uno tendrá la misma cantidad de información ($\frac{1}{K}$), y ningún significado por si solo. Debido a esto, si se pierde uno de los paquetes, no es necesario establecer ningún método para recuperarlo, ya que el siguiente que llegue puede reemplazarlo sin problema. La transmisión de un fragmento finaliza cuando se reciben K paquetes novedosos.

En ese momento, el rango de la matriz es K y, por tanto, se puede calcular su inversa C^{-1} para decodificar los paquetes del buffer de recepción, recuperando así la información original del fragmento (en notación matricial, $P = C^{-1} \cdot P'$). Tras ello, los K paquetes nativos se entregan, de manera simultánea, a la capa superior, que los recibe en el orden correcto. Con la recepción de un fragmento se desencadena el siguiente evento importante, el envío de un *Acknowledgment* (ACK). Hay que destacar que el nodo origen no interrumpe el envío de paquetes del fragmento actual hasta que no recibe el ACK, momento en el que se procede a eliminar todos los paquetes del buffer de transmisión, se actualiza el número de fragmento, comenzando con el envío del siguiente bloque de paquetes.

La Figura 2.5 muestra el diagrama (en el receptor) de dos fragmentos consecutivos. Al inicio se puede observar la recepción ideal de un fragmento (F_i), en la que todos los paquetes son novedosos (sus vectores de codificación $\overrightarrow{c}$ son linealmente independientes). Por tanto después de K transmisiones (cada una de ellas con un retardo medio de $\overline{\tau}$), el receptor tiene información suficiente para calcular la matriz inversa de los coeficientes, C^{-1}, y para recuperar la información original y enviarla a las capas superiores. En el caso del

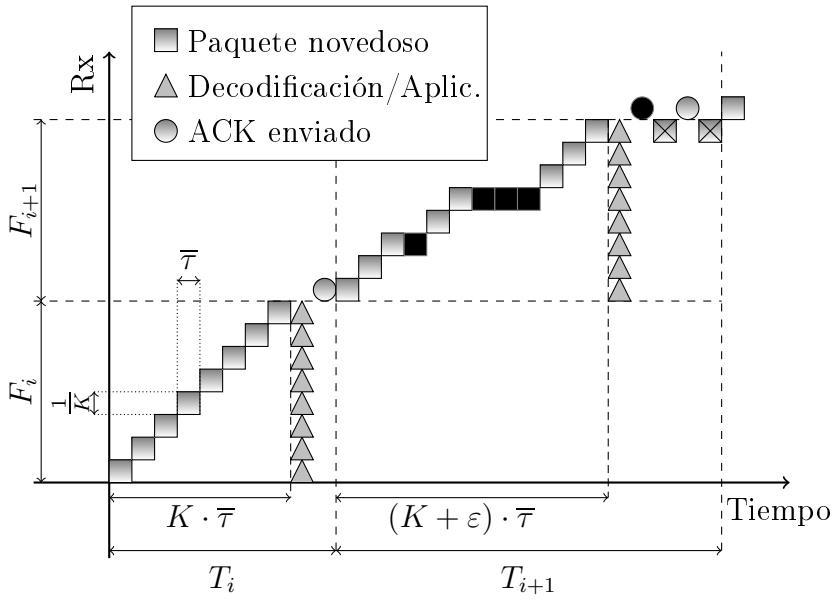


Figura 2.5: Eventos en recepción ($K = 8$)

segundo fragmento, se reciben varios paquetes no novedosos (representados como cuadrados negros en el diagrama). Estos serán descartados de forma automática, conduciendo a un rendimiento menor, ya que la información que aportan es nula, y la entidad receptora necesitará $K + \epsilon$ paquetes para decodificar el fragmento original, siendo ϵ el número de transmisiones adicionales necesarias para completar el fragmento. Finalmente, se envía al origen un ACK, para confirmar la correcta llegada del fragmento. Este ACK puede perderse, como en el diagrama y, por tanto, el origen no estará advertido del cambio de fragmento, por lo que que continuará enviando paquetes del mismo; estos son descartados, volviéndose a advertir al origen (mediante otro ACK) del cambio de fragmento, así hasta que se actualice la información en el origen.

Como se ha podido observar, el proceso confiere de una serie de propiedades a la transmisión. La primera de ellas es la de secuenciación. Los paquetes pueden llegar al destino en cualquier orden ya que, cuando se decodifican, la información original se recupera en secuencia correcta. Además, la pérdida de un paquete no es algo crítico, puesto que cada uno contiene la misma cantidad de información, siendo reemplazables entre ellos. El control de flujo se realiza de una manera muy liviana, ya que el origen envía paquetes constantemente hasta que el destino le confirma que puede parar. Por último, la codificación de los datos dota de mayor seguridad y robustez a la información.

Algunos de estos mecanismos son más propios del protocolo de transporte TCP, que se caracteriza por ser muy fiable, pero también por introducir una elevada sobrecarga, en especial, a la hora de ejercer el control de errores, de flujo y el secuenciamiento. Puesto que, como se ha comentado con anterioridad, NC implementa implícitamente todas estas

funcionalidades, se puede optar por sustituir el protocolo de capa de transporte por uno más ligero, y que penalice menos la transmisión de los datos, en este caso UDP.

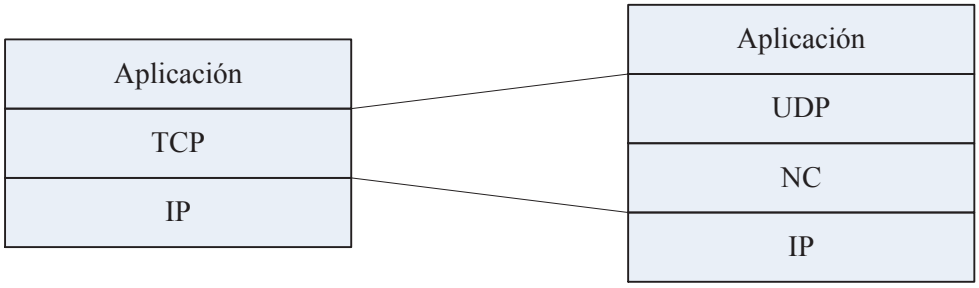


Figura 2.6: Esquema de cambio de arquitectura

Para realizar el mencionado intercambio de información y los procesos de codificación y decodificación se hace uso de la cabecera propietaria mostrada en la Figura 2.7. Se pueden apreciar dos partes bien diferenciadas: la primera, con una longitud fija de 9 bytes, contiene toda la información que debe ser intercambiada entre los nodos origen y destino (el tipo de paquete que se transmite, $Type$; el número de paquetes combinados, K; el número de bits utilizados para codificar cada coeficiente del vector, q; el número de fragmento; y los puertos UDP correspondientes). La segunda parte de la cabecera, que contiene el vector de coeficientes, tiene longitud variable. Es proporcional tanto a K como a q, y oscila entre 1 byte ($K = 2,\ q = 1$) y 192 bytes ($K = 255,\ q = 6$). La expresión final para la longitud de la cabecera completa es $9 + \left\lceil \frac{K \cdot q}{8} \right\rceil$ bytes.

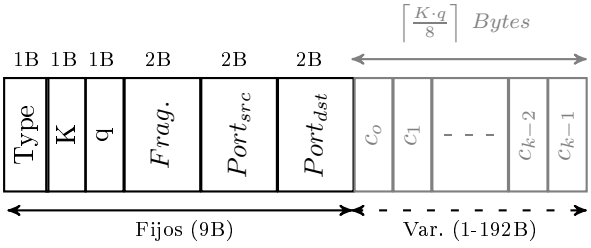


Figura 2.7: Formato de cabecera RLC

- El campo $Type$ identifica la clase de paquete que se está transmitiendo; como se verá en el Capítulo 4, se utiliza $Type = 0$ para un paquete de datos normal, $Type = 1$ para un ACK y $Type = 2$ para un ACK retransmitido tras una pérdida.
- El parámetro K indica la cantidad de paquetes codificados en cada fragmento. Su función es fundamental puesto que, como se verá en el Capítulo 5, la elección de este valor puede influir en gran medida en la cantidad de paquetes que se descartan por combinaciones lineales y, por consiguiente, en el rendimiento. Para valores bajos (matriz C de menor dimensión) la probabilidad de que la combinación sea linealmente

dependiente es mayor. Por otro lado, para valores altos, el coste computacional de las operaciones sobre la matriz C puede ser demasiado alto. A la vista de estos aspectos se tratará de llegar a una solución de compromiso entre un rendimiento aceptable y un coste computacional que no sea demasiado elevado.

- El campo q determina la cantidad de bits con los que se codifica cada coeficiente del vector. Guarda una relación inmediata con el tamaño del cuerpo finito utilizado para generar los coeficientes c_i de la forma: $GF(Q) = GF(2^q)$. El tamaño del cuerpo de Galois influye en el rendimiento de un modo similar al parámetro K. Cuando su valor es pequeño (por ejemplo: $q = 1$) los posibles valores para el coeficiente son reducidos, en este caso concreto serían "0" y "1". Sin embargo, para valores más altos, las librerías que manipulan estos elementos no son capaces de trabajar a la misma velocidad que con valores pequeños, por lo que el rendimiento se ve penalizado (más detalles en el Capítulo 5).

- El número de fragmento identifica el fragmento al que pertenece el paquete, permitiendo así a los nodos considerarlo o descartarlo en el caso de que llegue fuera del bloque correspondiente.

- Al final de la parte fija de la cabecera se dispone de las campos de Puerto UDP Origen ($Port_{src}$) y Puerto UDP Destino ($Port_{dst}$). Se hace necesario disponer de los datos de los puertos UDP en la capa de NC para poder distinguir la información proveniente de distintos flujos de datos (diferentes puertos origen y destino) y, así, poder gestionarlas por separado.

- Por último, se incluye el vector aleatorio de coeficientes c_i. Este campo de longitud variable es el que se extrae de la cabecera una vez el paquete ha llegado al destino y se introduce en la matriz C para comprobar si es linealmente independiente respecto a todos los anteriores y, por tanto, aporta información nueva.

Cabe destacar que los ACK tienen pequeñas diferencias. Para empezar, no se adjunta ningún vector en la cabecera, por lo que la parte variable no existe. Tanto el campo K como el campo q tienen su valor a "0" y el número de fragmento que contiene la cabecera es el del siguiente fragmento que espera recibir el destino.

Una vez explicado este escenario que ha servido para introducir los numerosos aspectos a tener en cuenta, se introduce un nodo intermedio entre origen y destino. En una primera aproximación, este nodo únicamente hace *forwarding* de los paquetes que le llegan del origen para entregarlos al destino. Sin embargo, para poder hablar propiamente de NC es necesario que la red (los nodos que la componen) realice labores de codificación. En la siguiente sección se explicará este proceso.

2.4.2. Recodificación en nodos intermedios

Como en el caso anterior se ilustrará con un ejemplo sencillo, que se recoge en la Figura 2.8. Esta vez el escenario consta de tres nodos, tratándose por tanto de un despliegue multi-salto. Los nodos origen y destino realizan el mismo proceso que en el ejemplo previo. Sin embargo, la pieza fundamental es el nodo intermedio. Este elemento no se limitará únicamente a labores de *forwarding*.

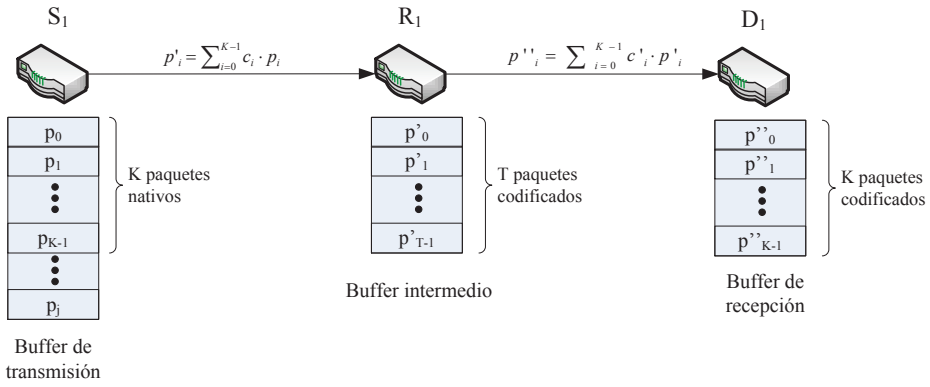


Figura 2.8: RLC en escenario de tres nodos

En este despliegue, R_1, a medida que recibe paquetes, los almacena y comprueba su independencia lineal, análogamente a como lo hace el nodo destino, D_1. Por tanto, es necesario que este nodo disponga también de una matriz para almacenar los vectores de coeficientes. Una vez que se ha alcanzado un número determinado de paquetes ($T \geq 2$) en el buffer, el nodo genera un nuevo paquete a partir de un vector aleatorio de coeficientes, combinando los paquetes codificados en el buffer. El nuevo paquete $p'' = \sum_{i=0}^{T-1} c'_i \cdot p'_i$ se pasa a las capas inferiores para que procedan a su envío al destino. Es importante recalcar que el vector de coeficientes que se introduce en la cabecera del paquete p'' no es el vector de coeficientes utilizado realmente $\overrightarrow{c'} = (c'_0, c'_1, \cdots, c'_{T-1})$, sino que es necesario operar $\overrightarrow{c'}$ con la matriz C del nodo intermedio para obtener el vector que se adjuntará en la cabecera (el desarrollo se detalla en el Capítulo 3). El destino trata este paquete como uno cualquiera y hace las comprobaciones pertinentes para incorporarlo, o no, al buffer de recepción.

Una vez se decodifica el fragmento nativo, el destino envía un ACK. Cuando el nodo intermedio recibe el ACK actualiza su propio número de fragmento, elimina los paquetes relativos al que acaba de ser decodificado y reenvía el ACK al nodo origen, S_1. En el caso de que se reciban paquetes fuera de fragmento (a causa de la pérdida del ACK o por paquetes remanentes del fragmento anterior), el nodo en cuestión (sea intermedio o destino) los descarta automáticamente, y envía un nuevo ACK para confirmar la entrega.

Cabe destacar que, en este punto, existen dos posibilidades de comportamiento. La primera implica que el nodo intermedio retenga el tráfico de paquetes codificados hasta que disponga de T paquetes en el buffer para comenzar a codificar. Este método funciona bien

con valores de T bajos, puesto que no implica una parada excesiva en el flujo de datos. La segunda opción es que el nodo intermedio haga *forwarding* de los paquetes codificados que reciba hasta que logre tener T paquetes en el buffer. A partir de ese momento únicamente transmitirá paquetes codificados por él. Su rendimiento es bueno con valores de T altos. Esta opción tiene una desventaja y es que, como se verá más adelante, puede provocar que se reciban una importante cantidad de paquetes duplicados, o con vectores de codificación linealmente dependientes.

A priori, la tasa de llegada de paquetes al destino mediante el método de recodificación en nodos intermedios no ofrece una mejora sustancial que justifique su coste computacional y de memoria. Sin embargo, es la piedra angular para lograr la mejora en el rendimiento que se está buscando. Para tratar de lograr el objetivo es necesario introducir una nueva técnica, el *overhearing*.

2.4.3. Overhearing, enrutamiento oportunista

El objetivo de esta técnica es aprovechar la característica broadcast del medio inalámbrico para aumentar el *throughput* de la transmisión. En un entorno de red mallada es posible que un nodo que no sea el destinatario de un paquete pueda "escuchar" la transmisión. Según el esquema tradicional, el nodo desecharía ese paquete. Sin embargo, con esta modificación un nodo puede recibir y manipular un paquete cuya dirección *Media Access Control* (MAC) destino no sea la suya propia. Así, parte de la información puede "adelantarse" a la ruta habitual. Cuando un nodo recibe un paquete mediante este método, el tratamiento que hace sobre él es el mismo que si se tratara de un paquete dirigido al propio nodo.

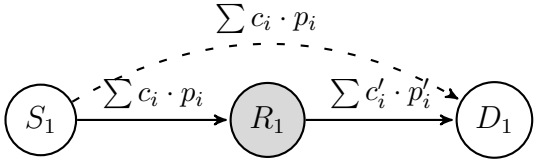


Figura 2.9: Escenario canónico de intra-flow NC

Combinado con la codificación de red en los nodos intermedios se puede lograr información adicional. Por un lado, un porcentaje de paquetes codificados una única vez, recibidos mediante el *overhearing*, p'_i, por otro lado, la información de los paquetes que se generan en los nodos intermedios, p''_i. Se puede observar un ejemplo en la Figura 2.9. En ella se aprecian con claridad las dos fuentes de información para el nodo D_1. La probabilidad de que D_1 "escuche" el paquete directo desde S_1, es pequeña debido a la distancia en el enlace directo entre ellos. A pesar de ello, si R_1 se limitara a hacer *forwarding* de ese mismo paquete, podría ser descartado puesto que D_1 ya habría recibido ese mismo paquete

por medio del *overhearing*. En contraste, si R_1 recodifica, la situación cambia, ya que se habrían recibido dos paquetes novedosos.

El *overhearing* no se limita únicamente al nodo destino, adelantando el instante de decodificación. También existe la posibilidad de que un nodo intermedio, configurado para recodificar paquetes, haga uso de esta técnica. Los paquetes que capte mediante el *overhearing* son tratados como cualquier otro paquete. Gracias a esto, se alcanza la cantidad de paquetes necesaria para empezar el proceso de recodificación en un tiempo menor. Este supuesto se representa en la Figura 2.10.

Por último, cabe comentar que en el Capítulo 5 se estudiará más en profundidad el rendimiento de los métodos tratados en esta sección, y se comparará con el escenario original.

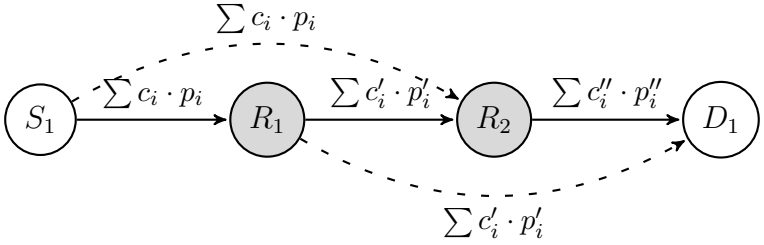


Figura 2.10: Escenario 4 nodos de intra-flow NC

2.5 Antecedentes

El término NC fue concebido originalmente en el año 2000 por *Ahlswede et al.* en [1]. En él se debate la idoneidad del paradigma clásico de *store-and-forward* en redes *Internet Protocol* (IP), sugiriendo que la integración de funcionalidades adicionales en los nodos intermedios puede llevar a sustanciosas mejoras en el rendimiento. Desde ese momento, varios trabajos han propuesto el uso de estos mecanismos para lograr tales mejoras y comunicaciones más fiables. Después de varios años de desarrollo, la comunidad científica acepta la división de las técnicas de NC en dos grupos; el primero de ellos está basado en la combinación de paquetes pertenecientes a distintos flujos en los nodos intermedios, siendo algunos de los ejemplos más populares *Opportunistic Coding in Practical Wireless Network Environment* (COPE) [3] y *Coding Applied To Wireless On Mobile Ad-Hoc Networks* (CATWOMAN) [4]. El segundo grupo promueve el uso de combinaciones lineales aleatorias de paquetes pertenecientes al mismo flujo; para descripciones más extensas se puede consultar [5] y [6].

Además, existe una tendencia adicional, enfocada a mezclar las dos técnicas mencionadas en una única operación de NC para, así, beneficiarse de las ventajas de cada uno

de ellas, como proponen I^2NC [7] y CORE [8]. En ellos se muestra que este acercamiento combinado mejora las soluciones individuales (tanto el *inter-flow* NC como el *intra-flow* NC).

Sin embargo, no existen demasiados trabajos que fomenten el uso de NC para mejorar el pobre rendimiento de TCP sobre redes inalámbricas con pérdidas. La causa es que habitualmente los estudios existentes no prestan atención a la capa de transporte y únicamente se centra en el mecanismo de NC por sí solo. Existen algunos textos que sí han lidiado con el análisis de estos casos [3], y coinciden en que la interacción entre NC y los algoritmos de control de la congestión de TCP puede ser bastante perjudicial para el rendimiento.

Centrando la atención en implementaciones concretas de NC, Chachulski et al. [9] proponen un nuevo protocolo, MORE, que combina un bloque de *Random Linear Network Coding* (RLNC) con enrutamiento oportunista. A pesar del hecho de que MORE mejora el rendimiento (en términos de fiabilidad y *throughput*) sobre WMN en comparación con otras soluciones oportunistas, ExOR [10], también presenta una serie de defectos, si se emplea con tráfico TCP ya que la transmisión de largos bloques puede, como se ha dicho anteriormente, provocar conflictos con los mecanismos de control de la congestión de TCP. Sundararajan et al. [6] realizan una aproximación distinta, y proponen algunas modificaciones en el protocolo anterior, añadiendo un mecanismo de reconocimiento de paquetes (ACK). Por último, Krigslund et al. en [8] combina la operación de COPE [3] con MORE [9], uniendo los acercamientos *intra* e *inter-flow* para explotar los beneficios de ambas técnicas.

Por otro lado, la popularidad que ha adquirido ns-3 recientemente ha fomentado la creación de diferentes trabajos que estudian la interacción entre NC y las WMN utilizando este herramienta de simulación. Por ejemplo, Yang Chi et al. propusieron *Yet Another Network Coding Implementation* (YANCI) [11], una versión simplificada de COPE. Sin embargo, los autores no consideran el borrado de paquetes en la red. Además, merece la pena resaltar la existencia de diferentes entornos de código abierto que admiten la posibilidad de enlazar ns-3 como una librería externa para simular NC sobre diferentes entornos de red, tales como NECO [12] o Kodo [13].

2.6 User Datagram Protocol (UDP)

UDP es un protocolo de la capa de transporte; es no orientado a la conexión, por lo que el intercambio de datagramas o mensajes se realiza sin establecer una conexión previa entre origen y destino. UDP no garantiza la entrega de los paquetes en orden y no dispone de un sistema de retransmisión. Si se desea disponer de estas funcionalidades es necesario que estén implementadas en otra capa. Esto permite una mayor velocidad en la comunicación, a cambio de perder fiabilidad en la transmisión. Los ámbitos donde este protocolo tiene mayor utilidad son escenarios donde haya un intercambio importante de

paquetes de conexión y desconexión, o éste no sea rentable, respecto a la información que se transmite (DHCP, BOOTP, DNS). Además, se utiliza para la transmisión de vídeo y audio en tiempo real, en las que los requisitos de retardo son muy estrictos, ya que en estos casos importa más que la información llegue en el momento adecuado que la pérdida puntual de uno de los paquetes.

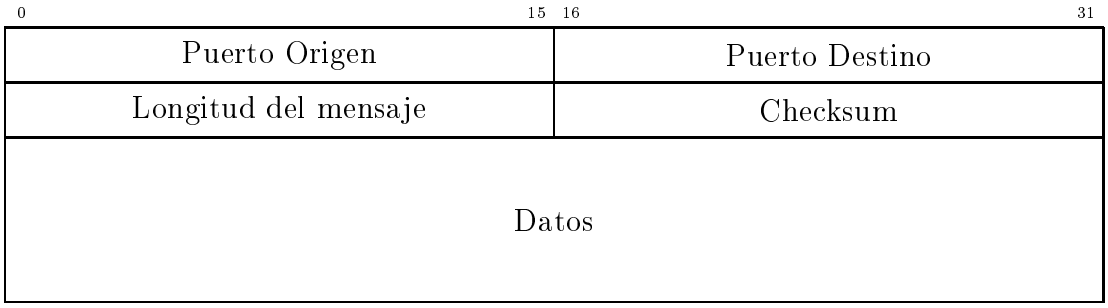


Figura 2.11: Cabecera UDP

Siguiendo con lo comentado anteriormente, el formato de cabecera UDP (Figura 2.11) es muy simple. La longitud total es de 8 bytes, repartidos de la siguiente forma:

- Puerto origen: campo de 16 bits que indica el puerto que origina la transmisión del mensaje. Junto con el puerto destino, es necesario para identificar la comunicación.
- Puerto destino: campo de 16 bits que indica el puerto del destino al que va dirigido el mensaje.
- Longitud del mensaje: indica el tamaño total en bytes del datagrama UDP completo, incluyendo cabecera y datos. La longitud de este campo es de 2 bytes.
- Checksum: este campo realiza una suma de comprobación de la combinación de una pseudo-cabecera IP, la cabecera UDP, los datos y un relleno de 0's para que el número de bits sea múltiplo de 16. En la Figura 2.12 se puede apreciar la totalidad de los datos sobre los que se aplica el checksum.

Como se ha comentado con anterioridad, este protocolo descarga ciertas responsabilidades hacia los protocolos de otras capas. Algunas de ellas son:

- Control de flujo: los paquetes a nivel de de transporte pueden llegar desordenados. En el caso concreto de NC, debido a que cada paquete tiene la misma cantidad de información y no está secuenciada, esta funcionalidad no implica ningún problema.
- Reconocimiento de paquetes: UDP no hace uso de ningún tipo de ACK para confirmar los paquetes. NC subsana este problema con el reconocimiento de paquetes por lotes, o fragmentos, de tamaño K.

- Mecanismos de retransmisión: este protocolo no plantea la retransmisión de un paquete que se haya perdido en la red. Para solucionar este problema, NC hace que un paquete perdido no sea importante, puesto que aporta la misma información que el siguiente.
- Mecanismos de control de la congestión: al igual que en punto anterior, el hecho de que se descarten paquetes de manera silenciosa cuando haya congestión en la red, no es un escollo para NC, que recupera la información que se pierde en esos paquetes con otros generados a partir del mismo fragmento.

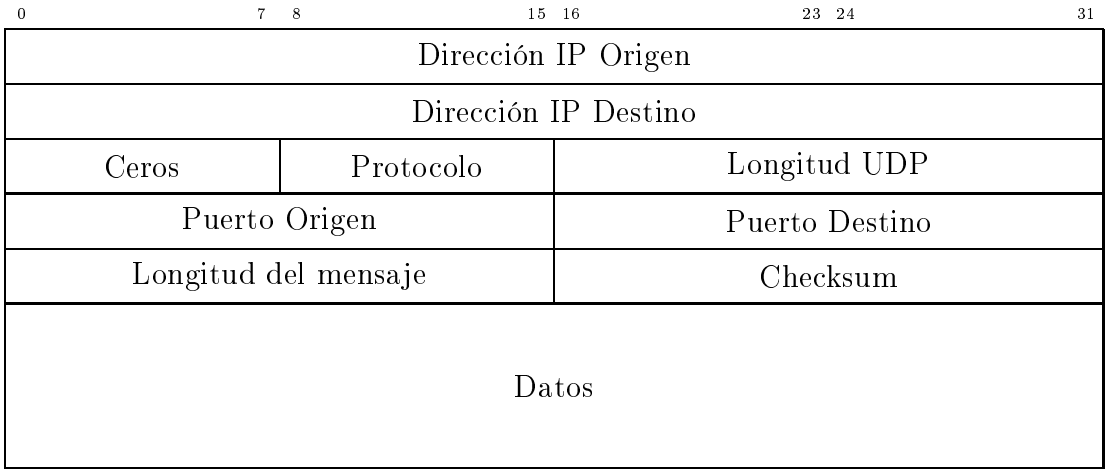


Figura 2.12: Datos para el cálculo del checksum

3

Teoría e Implementación

En este capítulo se presentan una serie de desarrollos teóricos básicos para comprender el funcionamiento y el rendimiento de las técnicas RLC. Además, se exponen las características del canal que se ha configurado en el simulador para llevar a cabo los experimentos. Con ello, se pretende brindar al lector los últimos elementos necesarios para comprender las características, tanto de las técnicas utilizadas, como los escenarios sobre los que se desarrollan las simulaciones del Capítulo 5.

3.1 Análisis del rendimiento

El escenario planteado para este estudio consiste en dos nodos, en el que uno de ellos transmite la información y el otro la recibe. El motivo de esta elección es que es uno de los escenarios sobre los que se va a trabajar en la campaña de simulaciones y que, además, no presenta una complejidad excesiva, por lo que se puede modelar de manera analítica. Se asume que el protocolo de capas inferiores utilizado es IEEE 802.11b.

3.1.1. Rendimiento bruto de RLC

El punto de partida de este estudio es el throughput que percibe la capa de RLC, que se corresponde con el ofrecido por el protocolo UDP. En este análisis se prescinde de la influencia de las capas inferiores, utilizando el throughput máximo ofrecido por UDP, $\overline{Thput_{UDP}}$, como referencia básica.

Para compensar el servicio no confiable proporcionado por UDP, la técnica requiere una serie de mecanismos para implementar ciertas funcionalidades que introducen *overhead*, como son las transmisiones en exceso o los paquetes de ACK enviados para confirmar los fragmentos.

El primero de ellos se corresponde con la transmisión de paquetes con vectores de coeficientes, $\overrightarrow{c}$ linealmente dependientes y que , por tanto, se descartan de forma automática al recibirse. Estas transmisiones pueden influir negativamente en el rendimiento, ya que se pueden considerar como inútiles (hablando en términos de aportar nueva información a la capa RLC). En [14], se determina la probabilidad de lograr una transmisión ideal de un fragmento (no se desecha ningún paquete) en función del parámetro K y del orden del cuerpo finito $GF(Q)$; esto es, $n = K$ paquetes novedosos tras K recepciones, Ecuación 3.1. Por otro lado, la Ecuación 3.2[1] muestra la *cumulative distribution function* (cdf) de la recepción de K paquetes novedosos tras n recepciones (siendo $n = N \geq K$). En este caso, $N - K$ representaría el número de paquetes en exceso que son descartados.

$$P_{K,q}(n = K) = \frac{q^{K^2}}{(q^K - 1)^K} \cdot \prod_{j=1}^{K} \left(1 - \frac{1}{q^j}\right) \tag{3.1}$$

$$P_{K,q}(n = N > K) = P_{K,q}(n = K) \cdot \left(\begin{bmatrix} N \\ N-K \end{bmatrix}_q + \sum_{t=1}^{N-K} (-1)^t \binom{N}{t} \begin{bmatrix} N-t \\ N-K-t \end{bmatrix}_q \right) \tag{3.2}$$

Una vez conocida la influencia del parámetro K sobre la cantidad de combinaciones lineales en los vectores de coeficientes, se puede obtener fácilmente el número de transmisiones en exceso (Ecuación 3.3). Este parámetro se utilizará para representar la relación entre el número de paquetes en exceso y el total de transmisiones, Ecuación 3.4, en la que ϵ representa un factor que penaliza el rendimiento por las posibles dependencias lineales.

$$E[P_{K,q}(n)] = \sum_{i=N-K}^{\infty} i \cdot P_{K,q}(i) \tag{3.3}$$

$$\epsilon = \frac{E[P_{K,q}(n)]}{K + E[P_{K,q}(n)]} \tag{3.4}$$

A esto hay que añadir que la correcta decodificación de un fragmento propiciará el envío de un ACK desde el destino a la fuente. El tiempo que tarda en transmitirse este paquete no debe considerarse como despreciable y, por tanto, tiene cierto impacto sobre el rendimiento final. Si se considera $\overline{\tau_{data}}$ como el retardo medio de un paquete de información, y $\overline{\tau_{ack}}$ como el tiempo medio requerido por un ACK, es posible obtener el parámetro ϵ_{ack}, que representa el factor de penalización asociado al envío de ACK's, tal y como se ve en la Ecuación 3.5.

[1] $\begin{bmatrix} m \\ n \end{bmatrix}_q = \frac{(q^m-1)(q^{m-1}-1)\cdots(q^{m-n+1}-1)}{(q^n-1)(q^{n-1}-1)\cdots(q-1)}$

$$\epsilon_{ACK} = \frac{\overline{\tau_{ack}}}{(K+\epsilon)\cdot\overline{\tau_{data}} + \overline{\tau_{ack}}} \tag{3.5}$$

A la vista de esto, se puede obtener el throughput que percibe la capa de aplicación, $\overline{Thput_{RLC}}$, como se puede observar en la Ecuación 3.6.

$$\overline{Thput_{RLC}} = \overline{Thput_{UDP}}\cdot(1-\epsilon)\cdot(1-\epsilon_{ack}) \tag{3.6}$$

3.1.2. Rendimiento sobre enlaces IEEE 802.11b

En el análisis previo se ha modelado el rendimiento del esquema RLC/UDP sobre escenarios genéricos. En este nuevo caso, se asume que entre los nodos hay un enlace físico 802.11b. Puesto que los canales inalámbricos son propensos a producir errores, es necesario tener en cuenta el efecto producido por las posibles pérdidas de paquetes. Por motivos de simplicidad se asume que el canal no tiene memoria y, por tanto, los errores producidos se pueden considerar como eventos aleatorios, sin correlación entre ellos.

Con el objetivo de combatir en la transmisión, el estándar 802.11 [15] define un esquema de retransmisión cuyo objetivo principal es proporcionar una técnica de recuperación frente a la pérdida de tramas. En resumen, el protocolo especifica un determinado número de retransmisiones que se realizan cuando la fuente no recibe un reconocimiento del receptor tras un cierto tiempo de espera. Antes de cualquier retransmisión, un procedimiento exponencial de *backoff* duplica el número de *slots* de la *Contention Window* (CW) y el nodo volverá a contender por el acceso al canal. El impacto de esta técnica se modela en la Ecuación 3.7, donde $\overline{\tau_i}$ representa el retardo medio entre dos paquetes consecutivos como función de i, el número de tramas que se han perdido entre dos transmisiones ($i=0$ significa que no ha habido ninguna pérdida), t_c se corresponde con el tiempo determinista de una transmisión, parámetro que se ha calculado en diversos trabajos (por ejemplo en [16]); R es el número máximo de retransmisiones permitidas por cada trama; y, finalmente, σ es el tiempo de *slot* (20 microsegundos en 802.11b).

$$\overline{\tau_i} = (i+1)\cdot t_c + \left[\left(16\cdot\sum_{j=0}^{i}2^{j} \mod (R+1)\right) - \frac{i+1}{2}\right]\cdot\sigma \tag{3.7}$$

Sólo se consideran errores debido a los efectos de propagación del medio (en este escenario particular, el impacto producido por colisiones o el efecto de terminal oculto no se tienen en cuenta), que quedan representados mediante la *Frame Error Rate* (FER) del canal. De este modo, el retardo medio entre recepciones consecutivas se puede obtener de la Ecuación 3.8, donde p representa la probabilidad de que una trama se pierda (en un canal sin memoria, se puede asumir que $p = FER$, suponiendo que la recepción de una trama no tiene ninguna influencia sobre la siguiente).

$$E[\overline{\tau}] = \sum_{i=0}^{\infty}(1-p)\cdot p^i \cdot \overline{\tau_i} \tag{3.8}$$

Basándose en este retardo, es posible calcular el throughput que perciben las capas superiores, como se puede ver en la Ecuación 3.9, donde L es la longitud de datos del paquete, en bytes.

$$\overline{Thput_{RLC_{802.11b}}} = \frac{L\cdot 8}{E[\overline{\tau}]}\cdot(1-\epsilon)\cdot(1-\epsilon_{ack}) \tag{3.9}$$

3.1.3. Recodificación de paquetes

Los nodos intermedios son capaces de codificar paquetes nuevos a partir de otros que ya estaban codificados previamente (Ecuación 3.10). Cuando se genera un paquete nuevo en el nodo intermedio (Ecuación 3.11) es necesario calcular el valor del nuevo vector de coeficientes. Tanto c_i como d_j son los coeficientes aleatorios que se generan para codificar los paquetes.

$$p' = \sum_{i=0}^{K-1} c_i \cdot p_i \tag{3.10}$$

$$p'' = \sum_{j=0}^{T-1} d_j \cdot p'_j = \sum_{j=0}^{T-1} d_j \left(\sum_{i=0}^{K-1} c_i \cdot p_i \right) \tag{3.11}$$

A la vista de esto, el nuevo vector que debe incluirse en la cabecera se puede expresar como indica la Ecuación 3.12.

$$\begin{pmatrix} v_0 & v_1 & \cdots & v_{T-1} \end{pmatrix} = \begin{pmatrix} d_0 & d_1 & \cdots & d_{T-1} \end{pmatrix} \cdot \begin{pmatrix} c_0^0 & c_1^0 & \cdots & c_{K-1}^0 \\ c_0^1 & c_1^1 & \cdots & c_{K-1}^1 \\ c_0^2 & c_1^2 & \cdots & c_{K-1}^2 \\ \vdots & \vdots & \vdots & \vdots \\ c_0^{T-1} & c_1^{T-1} & \cdots & c_{K-1}^{T-1} \end{pmatrix} \tag{3.12}$$

Siendo $\overrightarrow{v}$ el nuevo vector, $\overrightarrow{d}$ el vector aleatorio usado para combinar con los paquetes recibidos y c_i los coeficientes originales de los paquetes.

3.2 Canal

Esta sección resume brevemente el comportamiento del canal sobre el que se van a sustentar las simulaciones del Capítulo 5, que funciona de manera sencilla. Se establece una FER para cada enlace del despliegue (se puede realizar fácilmente mediante ficheros de configuración) y, a la hora de determinar si un paquete se entrega sin error en el destino, se genera un número aleatorio entre 0 y 1; si está por debajo del valor de FER, el paquete se entrega correctamente, en caso contrario, el paquete se pierde. Sin embargo, detrás de este sencillo procedimiento hay un modelo de canal bastante más complejo cuyo comportamiento queda recogido en [11].

El modelo de atenuación se basa en el método de Friis para el cálculo de la atenuación del canal (Ecuación 3.13). La capa física tiene cuatro estados, que determinan el estado del enlace. En función del estado y de la energía de la señal que transporta el paquete de información se determina si el paquete alcanza satisfactoriamente el destino.

$$P_l(d) = P_l(d_0) + n10log_{10}(\frac{d}{d_0}) \tag{3.13}$$

El valor n representa el parámetro exponencial de pérdidas. En el modelo utilizado este valor es igual a 3. d_0 es un valor de referencia de 1.0 m. El valor de $P_l(d_0)$ viene determinado por la Ecuación 3.14.

$$P_l(d_0) = \frac{P_t G_t G_r \lambda^2}{16\pi^2 d_0^2 L} \tag{3.14}$$

En esta última expresión P_t es la potencia transmitida, G_t y G_r las ganancias de las antenas en transmisión y recepción, λ la longitud de onda y L las pérdidas del sistema (en este caso 1).

4

Network Simulator

En este capítulo se introduce la herramienta principal de este proyecto, el simulador de redes ns-3. Se realiza una breve explicación de su historia, sus características principales y los componentes que se han utilizado durante la realización del proyecto. Por último, se desglosa el desarrollo realizado para implementar la técnica de *intra-flow* NC y simular los escenarios planteados, la problemática surgida con respecto a algunas ideas planteadas inicialmente y las soluciones adoptadas para subsanarla.

4.1 Características

El origen de ns-3 se remonta al año 1989, cuando Srinivasan Keshav desarrolla el simulador de red *REAL*. La primera versión, ns-1, fue concebida en el *Lawrence Berkeley National Laboratory* (LBNL), entre 1995 y 1997, por Steve McCanne, Sally Floyd, Kevin Fall, y otros colaboradores. Esta versión era conocida como LBNL *Network Simulator* y derivaba directamente de *REAL*. El núcleo del simulador estaba programado en C++ y los *scripts* de los escenarios de simulación en *Tool Command Language* (Tcl). Sun Microsystems, la Universidad de California, y la Universidad Carnegie Mellon fueron algunas de las organizaciones que realizaron importantes contribuciones al proyecto.

En el periodo de 1996-97 se inicia el desarrollo de una nueva versión, ns-2. Una de las principales variantes es la sustitución de Tcl por *Object Tcl* (OTcl), un dialecto de Tcl orientado a objetos, desarrollado por el *Massachusetts Institute of Technology* (MIT). Actualmente, ns-2 consta de aproximadamente 300000 líneas de código, a las que hay que sumar una cantidad similar de código generado por otros usuarios y que no está incluido en la distribución principal. El simulador se puede instalar sobre GNU/Linux, FreeBSD, Solaris, Mac OS X y versiones de Windows que soporten Cygwin.

Posteriormente, un equipo liderado por Tom Henderson (Universidad de Washington), George Riley (Instituto Tecnológico de Georgia), Sally Floyd, y Sumit Roy (Universi-

dad de Washington) obtuvo una serie de subvenciones de la Fundación Nacional de Ciencia de los Estados Unidos para implementar un sustituto para ns-2, que se llamó ns-3. El equipo colaboró con el proyecto Planete de INRIA, con Mathieu Lacage como líder de desarrollo de software, y formaron un nuevo proyecto de código abierto al que, posteriormente, se unieron otros desarrolladores de todo el mundo.

El desarrollo de ns-3 comenzó en julio de 2006. Se decidió abandonar por completo las compatibilidades con las versiones anteriores. El nuevo simulador se implementa con el lenguaje de programación C++. Gustavo Carneiro contribuyó al proyecto mediante una plataforma para generar *bindings* de Python (*pybindgen*), y el uso del sistema de compilación Waf.

La fecha de la primera versión, ns-3.1, fue junio de 2008. Posteriormente el software ha recibido actualizaciones de manera periódica hasta el día de hoy (mayo de 2014) en el que se dispone de la versión 3.19, que fue publicada en diciembre de 2013.

Se trata de un simulador de redes de eventos discretos desarrollado principalmente para su uso en ámbitos de investigación y educación [17]. El objetivo del proyecto ns-3 es desarrollar un entorno abierto de simulación para investigación en el ámbito de las redes. ns-3 es software libre, bajo una licencia GNU GPLv2, y está disponible para que cualquiera pueda hacer uso del código fuente ya desarrollado, o para contribuir con módulos adicionales. Este punto concreto es el que dota de un fuerte interés al proyecto. Trata de conjugarse con las necesidades investigadoras en redes modernas y, además, anima a la comunidad a contribuir, revisar y validar la plataforma. La filosofía de software libre, unida a la facilidad para sumar aportaciones que mejoren el entorno de simulación hacen que la acogida por parte de los usuarios sea muy buena, tanto en el ámbito académico como en el de investigación.

El núcleo de ns-3 soporta el estudio de redes IP y de otras que no lo sean. Sin embargo, la mayor parte de los usuarios se centran en la simulación de entornos inalámbricos/IP, utilizando modelos para las capas 1 y 2 tales como: Wi-Fi, WiMAX y LTE. Además, incluye una amplia variedad de protocolos de enrutamiento estático y dinámico, como OLSR y AOVD.

ns-3 incluye un planificador de tiempo real para facilitar la interacción del simulador con sistemas reales. Gracias a esto, los usuarios pueden enviar y recibir paquetes generados en ns-3 sobre dispositivos de red reales.

En la Figura 4.1 se muestra la jerarquía de la versión de ns-3 utilizada, la 3.13. En la parte superior se encuentra el nivel de *Test*. Su cometido principal es el de permitir al usuario establecer las condiciones en que se desarrollará la simulación, así como determinar los escenarios sobre los cuales se realizará el experimento. Por debajo de éste, se encuentra el nivel *Helper*, que incluye una serie de módulos que facilitan la interacción entre el usuario y el núcleo de la simulación, al determinar la topología y los parámetros a simular. Así,

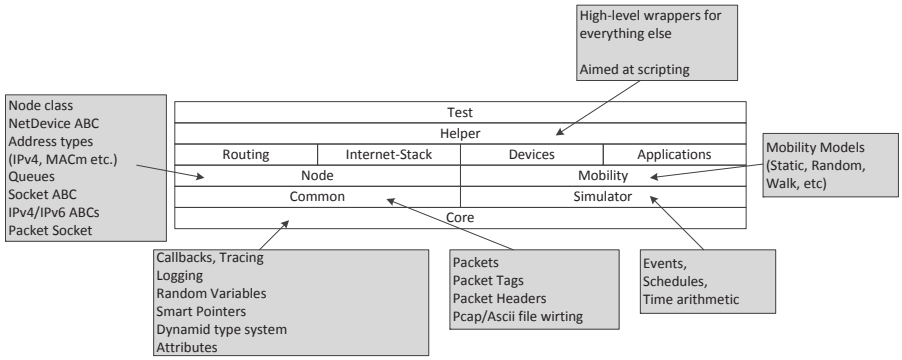


Figura 4.1: Jerarquía de organización de ns-3

se encarga de invocar, de manera automática y con los valores adecuados, a los elementos que conforman el escenario.

Justo debajo del nivel *Helper* aparecen los módulos que contienen los elementos sobre los que actúa capa. Algunos ejemplos son: los diferentes tipos de enrutamiento, la pila de protocolos de internet, la variedad de aplicaciones disponibles para la generación de datos o los tipos de dispositivos que se pueden incluir. Estos módulos representan clases que, en su mayoría, heredan de otras más genéricas, que se encuentran por debajo en el esquema de la Figura 4.1. Modelan una serie de elementos con características generales que pueden dar pie a multitud de entidades más complejas mediante la instanciación de algunos de sus parámetros. Estos módulos son independientes del tipo de red y de dispositivo. Por ejemplo, las clases de tipo *"node"* contienen atributos como el tipo de dirección (IPv4, MAC etc.), sockets para una comunicación TCP, las características de una posible cola, etc.

Como penúltimo nivel está la capa que contiene las herramientas propias del simulador (eventos, planificadores, temporizadores etc.) y la abstracción más genérica de los objetos (*packets*) que se utilizarán en las simulaciones. Finalmente se encuentra el núcleo del simulador que cuenta con las herramientas más básicas y fundamentales (punteros inteligentes, callbacks, variables aleatorias etc.) para conformar todo lo anteriormente mencionado.

Como se puede observar, se trata de una jerarquía hereditaria (propia de la programación orientada a objetos) en la que los niveles inferiores proporcionan las herramientas para generar las capas que están por encima.

4.2 Componentes

ns-3 cuenta con una cantidad muy grande de módulos de código fuente en C++ interconectados entre sí, que implementan las funcionalidades que pueda necesitar el usuario.

Para este proyecto, se ha trabajado especialmente con componentes relativos a UDP e IP. Para las capas física y MAC se han utilizado módulos de 802.11 y, a nivel de aplicación, una serie de ficheros para generar datos de una manera sencilla y flexible.

A continuación, se procede a desgranar cada uno de estos componentes.

4.2.1. Aplicación

Los módulos relativos a este nivel implementan las funcionalidades necesarias para enviar y recibir información. No se pretende realizar un estudio detallado de ninguna aplicación concreta, sino que el proyecto se centra en el rendimiento de la capa de NC. A continuación se hace una breve explicación de los módulos que se han empleado:

- **OnOffApplication**: Se trata de un generador de tráfico hacia un único destino, siguiendo un patrón on-off. Sus principales parámetros son: *onTime* (variable aleatoria que determina el periodo de tiempo que la aplicación está generando tráfico), *offTime* (se alterna con *onTime*, y se corresponde con el tiempo que la aplicación no enviará datos). El tráfico generado es de tipo *constant bit rate* (cbr) y viene determinado por los parámetros *cbrRate*, velocidad a la que la capa de aplicación genera los datos, y *pktSize*, que determina el tamaño de los paquetes generados por el nivel de aplicación. Por último, cabe destacar también el parámetro *maxBytes*, que limita la cantidad total de datos que se generarán.
- **PacketSink**: Esta aplicación complementa a *OnOffApplication*. Su cometido es recibir datos en una dirección y un puerto determinados. Sus principales parámetros son: *local* (determina la dirección en la que se reciben los datos), *totalRx* (monitoriza la cantidad de bytes recibidos por la aplicación hasta ese instante), y *socketList* (contiene una lista de todos los sockets en comunicación con la aplicación). Además, se ha creado la posibilidad de realizar un callback a esta clase (*rxTrace*), para permitir el traceado de información sobre los datos que se van recibiendo para su posterior estudio.

4.2.2. Transporte

Como se ha comentado previamente, el protocolo utilizado en la capa de transporte ha sido UDP. ns-3 dispone de una serie de ficheros fuentes para implementar la funcionalidad de este protocolo. A continuación se pasa a describir cada una de ellas.

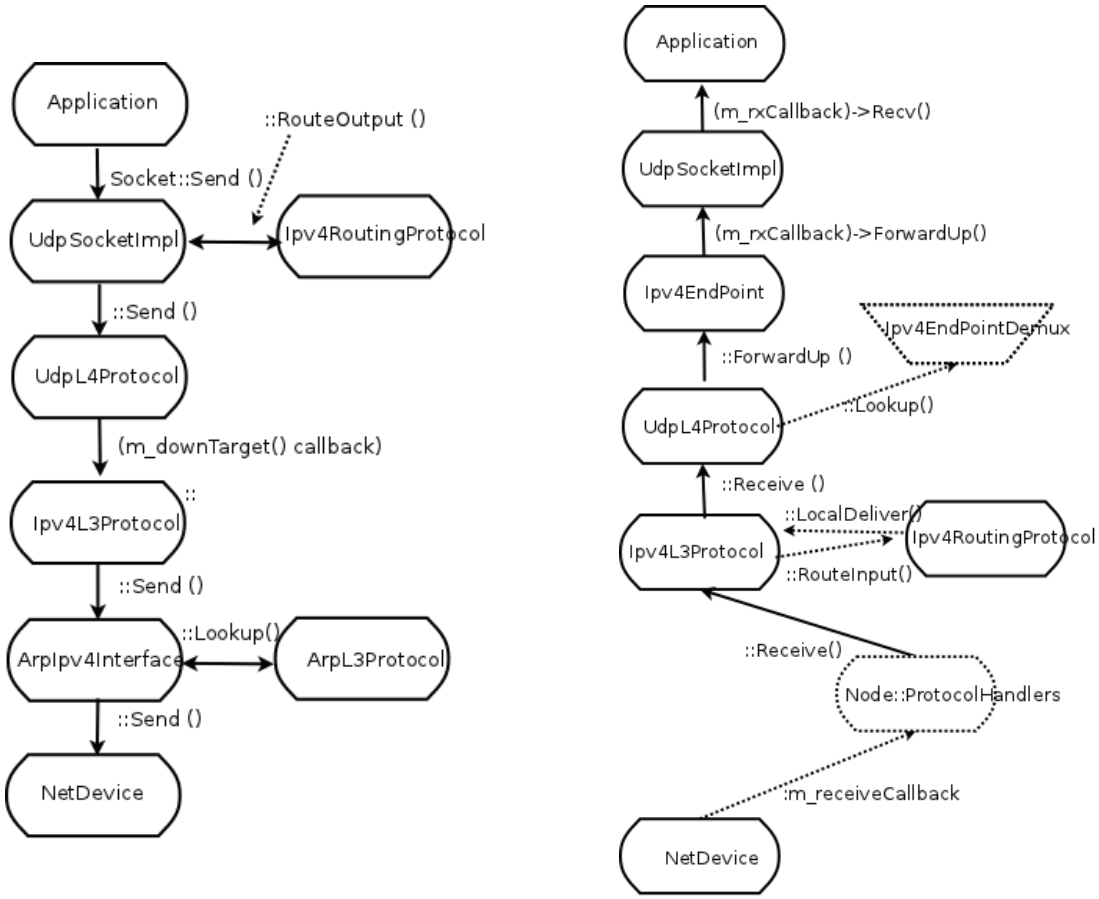
- **UdpSocket**: Esta clase existe simplemente para almacenar los atributos de socket UDP que pueden ser utilizados en diferentes implementaciones. También sirve para declarar una API específica para crear sockets UDP.

- **UdpSocketImpl**: Es una subclase de *UdpSocket*, que proporciona una interfaz para interactuar con los socket UDP implementados.
- **UdpSocketFactory**: Define la API para crear instancias de sockets UDP.
- **UdpSocketFactoryImpl**: Subclase de *UdpSocketFactory*, que implementa la API para crear sockets UDP.
- **UdpHeader**: Define la estructura de la cabecera UDP que se añadirá a los paquetes al pasar por la capa 4 (transporte). Define también los métodos específicos necesarios para la manipulación de los elementos de la misma. Algunos de ellos son: obtención y establecimiento de los puertos de la comunicación (*SetDestinationPort*, *GetDestinationPort*, *SetSourcePort*, *GetSourcePort*), y generación y comprobación del checksum de la cabecera (*InitializeChecksum*, *IsChecksumOk*). El resto los hereda de la clase *Header*.
- **UdpL4Protocol**: Define el comportamiento y la implementación del protocolo UDP. Contiene métodos para enviar y recibir información de nivel de transporte, funcionando como módulo intermedio entre el socket UDP y el protocolo de capa 3 (en este caso IP). Declara un callback para permitir interrumpir el paso de los datos de la capa de transporte a la de red, para introducir el protocolo NC entre ellas.

4.2.3. Red

En el nivel de red, se utilizará el protocolo IP, que es el más ampliamente utilizado en la actualidad para redes de usuario, puesto que es el protocolo que utiliza Internet. Al igual que UDP, su comportamiento está implementado en una serie de ficheros fuente que se describen a continuación.

- **Ipv4EndPoint**: Esta clase representa una conexión internet extremo-extremo a través de una tupla de cuatro campos (puertos origen y destino y direcciones origen y destino). Gracias a ellas es posible realizar una búsqueda de la conexión a la que se corresponde un paquete. Se utiliza dentro de la clase *UdpL4Protocol*.
- **Ipv4**: Define una API para la manipulación de diversos elementos del protocolo como la tabla de rutas (*Ipv4RoutingProtocol*), las interfaces de red de los dispositivos, añadiendo o eliminando direcciones IP de las mismas (*Ipv4InterfaceAddress*), y los atributos del protocolo.
- **Ipv4Header**: Conforma la cabecera IP de los paquetes a transmitir. Permite la extracción, establecimiento y modificación de cada uno de sus campos a través de una serie de métodos específicos. También permite realizar ciertas operaciones con los campos (comprobar el checksum, comprobar si hay más fragmentos por transmitir).



(a) Envío (b) Recepción

Figura 4.2: Esquemas de recorrido de los paquetes en ns-3

Al igual que *UdpHeader*, hereda de la clase *Header* y contiene sus funciones básicas para la manipulación de cabeceras.

- **Ipv4L4Protocol**: Se trata de una clase abstracta para que los protocolos de capa 4 (transporte) que utilicen IP puedan heredar de ella y tener acceso a parte de las funcionalidades del nivel de red.
- **Ipv4L3Protocol**: Implementa el protocolo de red IPv4. Proporciona API's para el manejo del enrutamiento, envío y recepción de paquetes, y los métodos necesarios para la manipulación de todos los elementos del protocolo. Dispone de dos fuentes para analizar los paquetes tanto en transmisión como en recepción. Además, se ha añadido un callback para interceptar la entrega desde este nivel (en recepción) a la capa superior (transporte) y así poder introducir NC como protocolo intermedio.

La Figura 4.2 muestra el trayecto que siguen los paquetes a través de los módulos de ns-3 (desde la capa de aplicación hasta la de red) y las interfaces entre ellos.

Existen una serie de módulos adicionales que se encargan de controlar el enrutamiento, las interfaces y otros. Para más información sobre ellos se recomienda consultar la documentación de ns-3 [17] y el repositorio de código [18].

4.2.4. Enlace y físico

En las capas inferiores (MAC y física) se ha establecido como protocolo de comunicación 802.11b. La implementación se divide en diversos ficheros que definen tanto las funcionalidades de los niveles físico y MAC. A continuación se describen brevemente las más importantes en el marco de este proyecto.

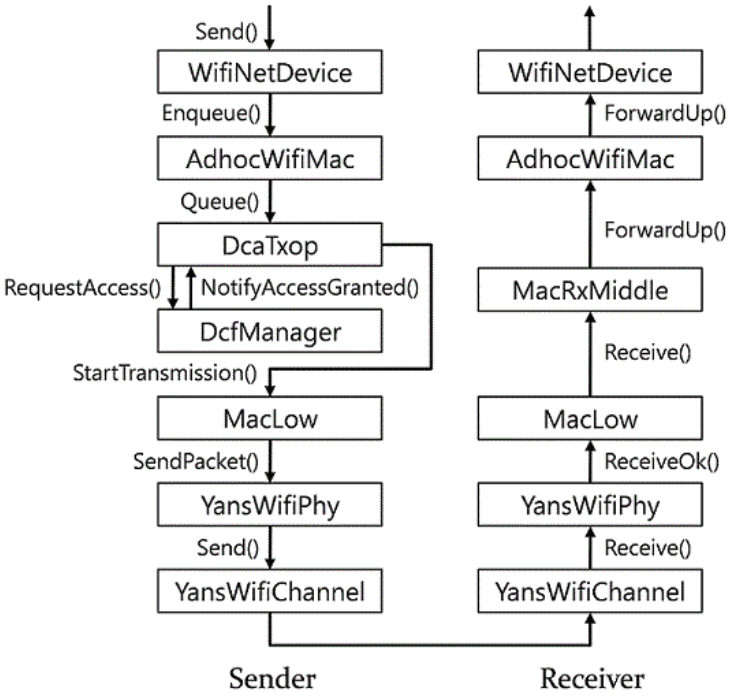


Figura 4.3: Wi-Fi en ns-3

- **WiFiMacQueue**: Esta clase implementa el comportamiento de la cola que administra el paso de paquetes desde el nivel MAC al físico. Modela el temporizador que especifica el estándar 802.11. Si se determina que el paquete debe salir de la cola, el método *Dequeue* se encarga de entregarlo a la capa física.
- **WiFiMacHeader**: Al igual que con los casos anteriores, este módulo implementa la cabecera del nivel MAC con cada uno de sus campos. Además, incluye una serie de funciones para manipularlos (*set's* y *get's*) y para realizar las pertinentes comprobaciones cuando llega un paquete (identificación del tipo, funciones de *Quality of Service* (QoS), control del proceso de autenticación etc.).

- **WiFiMac**: Este módulo describe el comportamiento del nivel MAC de 802.11. Encapsula todas las funcionalidades del protocolo, tanto de bajo nivel (DCF, EDCA, etc.) como de alto nivel (máquinas de estado para la asociación o desconexión de los dispositivos etc.)
- **YansWiFiPhy**: Contiene el modelo de nivel físico de 802.11, descrito en *"Yet Another Another Network Simulator"* [19]. Su comportamiento depende de las condiciones seleccionadas en otros módulos (*PropagationLossModel* y *PropagationDelayModel*, incluidos en *YansWiFiChannel*). Incluye *callbacks* para poder monitorizar la recepción y envío de paquetes en la capa física.
- **WiFiPhy**: Contiene diversas clases relativas al nivel físico de 802.11. Controla las notificaciones de eventos de la capa 1 y realiza la implementación del protocolo.

En la Figura 4.3 se muestra la sucesión de módulos que conforman el nivel 802.11 (Wi-Fi). Se observan algunos ficheros, de menor importancia o con funcionalidades muy concretas, que no han sido comentados anteriormente; para una correcta documentación sobre ellos el lector puede acudir a la bibliografía del simulador [17].

Una vez descritos los módulos de ns-3 que van a conformar todos los aspectos relativos a la transmisión de los datos que no son inherentes a NC, se procederá a describir, en la siguiente sección, el desarrollo necesario para implementar las funcionalidades de esta nueva capa.

4.3 Desarrollo en el marco ns-3

Esta sección concentra lo que ha sido el núcleo principal de trabajo de este proyecto. Aquí se expondrá el desarrollo llevado a cabo para implementar la capa de NC y sus funcionalidades en el entorno del simulador ns-3.

La estructura en que se han dispuesto los distintos componentes de NC se basa en un fichero que determina el formato y funciones de manipulación de la cabecera, y otro módulo describiendo el protocolo y su comportamiento. De este modo, NC se constituye con dos ficheros de cabeceras (*inter-flow* e *intra-flow* NC) y dos ficheros que describen ambos protocolos. Además, se incluyen un par de módulos extra para implementar el comportamiento del buffer en *inter-flow* NC, y otra clase que hereda de *Ipv4L4Protocol*. Más adelante, en esta misma sección, se describirán en detalle los componentes mencionados.

Además de la implementación del protocolo, es necesario brindar al usuario la opción de configurar el escenario y los parámetros con los que se desarrollará la simulación. Para ello se ha realizado un sistema de ficheros de configuración que permite interactuar con

el código fuente C++ encargado de establecer los parámetros del escenario de manera sencilla. Para ello, se cuenta con tres ficheros.

El primero de ellos determina la localización de los nodos en el escenario y la función de cada uno de ellos. Consta de una tabla que indica la posición de cada nodo que, además, se emplea para establecer si un nodo está capacitado para recibir paquetes (campo RX a 1), si envía datos a otro nodo (campo TX con el valor del nodo al que se transmite) o si debe realizar tareas de *forwarding* (campo FW a 1). El campo CR determina si el nodo recodifica paquetes en la modalidad de *inter-flow* NC. En la Tabla 4.1 se puede apreciar la disposición de cada campo en el fichero.

Tabla 4.1: Fichero de configuración espacial de los nodos

Nodo	X	Y	Z	TX	RX	CR	FWD
1	0	0	0	3	0	0	0
2	9	0	0	0	0	1	1
3	18	0	0	0	1	0	0

A la vista de los datos contenidos en la Tabla 4.1, la disposición final del escenario sería la mostrada en la Figura 4.4.

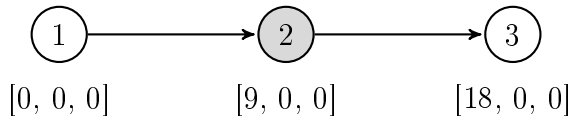


Figura 4.4: Disposición espacial de los nodos

El siguiente fichero se utiliza para determinar las condiciones del canal. Configura el valor de la FER de los enlaces entre dos nodos. La representación es como la mostrada en la Tabla 4.2. Si el valor del campo es un 0, la FER es 0 y, por tanto, el enlace sería ideal, sin que se produzcan pérdidas. En el caso de que sea 1, la tasa de error del enlace es 1 y la comunicación sobre él es inviable. Por último, si el campo tiene el valor 5 ó 6, obtiene el valor de la FER a partir de los parámetros *FER* y *FER_STATIC*, del fichero *network-coding-scenario.conf*, respectivamente. Esto permite modelar tanto los enlaces adyacentes (5) como los enlaces directos entre nodos no contiguos (6), para estudiar las técnicas de escucha oportunista.

Tabla 4.2: Fichero de configuración del canal

Nodo	1	2	3
1	1	5	6
2	5	1	5
3	6	5	1

La Figura 4.5 muestra el resultado de utilizar la Tabla 4.2 como configuración del canal.

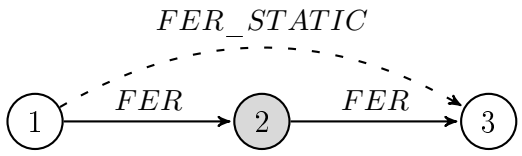


Figura 4.5: Configuración de los enlaces del despliegue

El último fichero de configuración es el correspondiente a la tabla de rutas. Indica cuál es el siguiente salto que tiene que utilizarse para que un paquete alcance el destino deseado. En la Tabla 4.3 se puede apreciar el formato.

Tabla 4.3: Fichero de configuración de rutas

NodoID	Destination	Nexthop	Interface	Metric
0	1	1	1	0
0	2	1	1	0
1	0	0	1	0
1	2	2	1	0
2	0	1	1	0
2	1	1	1	0

El campo *Destination* indica el destino. *Nexthop* determina el nodo al que debe ser enviado el paquete para alcanzarlo. La columna *Interface* permite conocer la interfaz del dispositivo por la que debe reenviarse el paquete. Por último, *Metric* permite determinar los costes en los enlaces para otras aplicaciones.

Cabe destacar que el valor de NodoID no coincide con el identificador de nodo. NodoID se inicia en 0 y el identificador en 1, por lo que para hacer la equivalencia entre ellos hay que sumar una unidad a NodoID.

Por otro lado, también interesa poder recoger los resultados de las simulaciones. Para ello se han desarrollado métodos que facilitan la generación de ficheros de trazas con los resultados. A partir de ellos, se pueden generar las figuras que se mostrarán en el Capítulo 5. Existen dos clases de archivos de trazas. El primero de ellos tiene un formato resumido, que contiene información sobre cada una de las simulaciones de manera general. En la Tabla 4.4 se puede observar un ejemplo de este tipo de fichero y, a continuación, se definen cada uno de los campos que lo forman.

- **No.**: Indica el número de la simulación cuando se programan múltiples repeticiones del experimento.

- **K**: Se corresponde con el número de paquetes nativos que irán codificados en cada fragmento.
- **q**: Determina el número de bits con los que se codifica cada coeficiente que multiplica a los paquetes nativos. Define el tamaño del cuerpo de Galois utilizado para generar los coeficientes. $GF(Q) = GF(2^q)$.
- **FER**: Indica la FER entre nodos adyacentes en el despliegue.
- **FER_2**: Define la FER entre nodos no adyacentes (enlace directo).
- **Thput**: Muestra el throughput, a nivel de aplicación, que se ha obtenido en la simulación, a partir de la transmisión completa de todos los paquetes.
- **Packets**: Número de paquetes nativos generados por la fuente.
- **Tx**: Número de paquetes de NC que se han transmitido durante la simulación. Incluye los paquetes transmitidos por la fuente y por los nodos intermedios.
- **Rx**: Número de paquetes de NC que se han recibido en el destino.
- **Time**: Tiempo total necesario para transmitir todos los paquetes.
- **Bytes**: Número total de bytes de datos transmitidos.
- **AvgDelay**: Retardo medio (en ms) entre la recepción de fragmentos.
- **VarDel**: Varianza en el retardo (en ms^2) de todos los fragmentos.
- **AvgInv**: Tiempo medio (en ms) para calcular la inversa de la matriz de coeficientes al decodificar los paquetes.
- **VarInv**: Varianza (en ms^2) del tiempo necesario para el cálculo de la inversa.
- **AvgRank**: Tiempo medio (en ms) para calcular el rango de la matriz de coeficientes cuando llega un nuevo paquete al destino.
- **VarRank**: Varianza (en ms^2) del tiempo necesario para el cálculo del rango.

Tabla 4.4: Formato de la primera versión del fichero de traza resumida

No.	K	q	FER	FER_2	Thput	Packets	Tx	Rx	Time	Bytes	AvgDelay	VarDel	AvgInv	VarInv	AvgRank	VarRank
1	8	2	0.00	0.20	1853.74	1024	2160	1180	12.951	2985984	0.1019	0.0005	0.0174	0.000084	0.0159	0.0059
2	8	2	0.00	0.20	1866.60	1024	2151	1097	12.715	2985984	0.1001	0.0004	0.0173	0.000105	0.0139	0.0051
3	8	2	0.00	0.20	1860.55	1024	2155	1089	12.830	2985984	0.1010	0.0003	0.0168	0.000078	0.0150	0.0038

Tabla 4.5: Formato de la segunda versión del fichero de traza resumida

No.	Node	FER	FER_2	PktLen	q	K	Thput	TX_{app}	RX_{app}	TX_{nc}	RX_{nc}	AvgDel	VarDel	AvgRank	VarRank	AvgInv	VarInv
1	0	0.00	0.20	1455	1	64	0.000	1024	0	1120	0	114.5867	1.34e+10	0.0000	0.00e+00	0.0000	0.0e+0
1	1	0.00	0.20	1455	1	64	0.000	0	0	1120	0	0.0000	0.00e+00	0.0000	0.00e+00	0.0000	0.0e+0
1	2	0.00	0.20	1455	1	64	2.876	0	1024	0	1344	3.0691	9.21e+06	0.1040	1.16e-01	0.3212	0.0e+0
2	0	0.00	0.20	1455	1	64	0.000	1024	0	1134	0	90.3954	8.27e+09	0.0000	0.00e+00	0.0000	0.0e+0
2	1	0.00	0.20	1455	1	64	0.000	0	0	1138	0	0.0000	0.00e+00	0.0000	0.00e+00	0.0000	0.0e+0
2	2	0.00	0.20	1455	1	64	2.816	0	1024	0	1365	3.0806	9.21e+06	0.1043	5.43e-02	0.1996	0.0e+0
3	0	0.00	0.20	1455	1	64	0.000	1024	0	1134	0	82.7485	6.92e+09	0.0000	0.00e+00	0.0000	0.0e+0
3	1	0.00	0.20	1455	1	64	0.000	0	0	1089	0	0.0000	0.00e+00	0.0000	0.00e+00	0.0000	0.0e+0
3	2	0.00	0.20	1455	1	64	2.877	0	1024	0	1295	3.1689	1.02e+07	0.1222	3.21e-01	0.2353	0.0e+0

Posteriormente se realizó una segunda versión del formato de traza resumida para disponer de más información del comportamiento de cada uno de los nodos de la simulación por separado, tal y como se muestra en la Tabla 4.5. En este caso cada línea no representa una simulación, sino el comportamiento de un nodo concreto en un experimento.

A continuación se explica el significado de los campos que no estaban presentes en la anterior versión de la traza:

- **Node**: Determina el nodo del que se está mostrando la información.
- **PktLen**: Cantidad de datos en cada paquete codificado; viene determinado por K y q.
- $\mathbf{TX_{app}}$: Número de transmisiones a nivel de aplicación que ha realizado el nodo.
- $\mathbf{RX_{app}}$: Paquetes recibidos a nivel de aplicación en el nodo.
- $\mathbf{TX_{nc}}$: Cantidad de transmisiones de paquetes codificados (NC) que ha realizado el nodo. Pueden tratarse de simples retransmisiones de un paquete o envío de un paquete recodficado en el propio nodo.
- $\mathbf{RX_{nc}}$: Número de paquetes de NC que se reciben en el nodo (sea un paquete novedoso o no).

A pesar de los detalles que incluyen estos ficheros, en ciertas ocasiones no es suficiente para realizar un estudio profundo del comportamiento del protocolo. Para ello, se ha desarrollado otro tipo de traza cuyo cometido es el de poder realizar un estudio temporal y pormenorizado de los eventos que ocurren durante la simulación. En la Tabla 4.6 se puede observar un ejemplo.

Tabla 4.6: Formato de traza temporal

Tiempo	Estado	NodoID	IpOrigen	IpDestino	Long	NºFragmento	UID
20.144633	0	0	10.0.0.1	10.0.0.3	1455	0	490689
20.145943	6	1	10.0.0.1	10.0.0.3	1455	0	490673
20.145943	2	2	10.0.0.1	10.0.0.3	1455	0	490673
20.145943	8	2	10.0.0.1	10.0.0.3	1455	0	490673
20.146327	7	1	10.0.0.1	10.0.0.3	1455	0	490692
20.147637	2	2	10.0.0.1	10.0.0.3	1455	0	490685
20.149631	2	2	10.0.0.1	10.0.0.3	1455	0	490692

En este formato, cada línea de código está asociada a un evento, relacionado con un paquete. El significado de cada campo se resume a continuación:

- **Tiempo**: Muestra el instante de tiempo en el que se ha producido el evento.
- **Estado**: Este campo indica el tipo de evento que ha ocurrido. Puede puede tratarse del envío de un paquete en la fuente, o la recepción de un ACK. Para más documentación sobre este campo se recomienda consultar el código fuente.
- **NodoID**: Determina el identificador del nodo donde se produce el evento.
- **IpOrigen**: Indica la dirección IP del origen del paquete.
- **IpDestino**: Indica la dirección IP del destino del paquete.
- **Long**: Número de bytes de datos que contiene el paquete de NC.
- **Nº Fragmento**: Número de fragmento dentro del flujo de transmisión del paquete que provoca el evento.
- **UID**: Identificador del paquete. Su utilidad es importante a la hora de identificar un paquete en el flujo temporal.

Una vez explicado el formato de las trazas, se pasa a detallar el desarrollo de los ficheros de código fuente C++ que se han desarrollado para la simulación del comportamiento de NC.

En primer lugar existen tres módulos implementados previamente para llevar a cabo las funcionalidades de *inter-flow* NC. Se describirán brevemente, puesto que no conforman el objetivo principal del proyecto:

- **InterFlowNetworkCodingHeader**: Describe la estructura de la cabecera de este tipo de NC, e implementa métodos específicos para su manipulación.
- **InterFlowNetworkCodingProtocol**: Constituye el núcleo central dela implementación. Incorpora cada uno de los métodos necesarios para el correcto funcionamiento del protocolo descrito en la Sección 2.3.
- **InterFlowNetworkCodingBuffer**: Uno de los elementos fundamentales de inter-flow NC es el buffer que mantiene los paquetes listos, a la espera de una oportunidad de codificación. Debido a su importancia en el proceso, dispone de un módulo completo para la descripción detallada de su funcionamiento.

Una vez vista la primera de las técnicas de NC, se procede a examinar la implementación de la otra modalidad, *intra-flow* NC. Al tratarse dela parte principal del trabajo, se detallará en profundidad su estructura y los métodos necesarios para lograr el comportamiento adecuado para realizar los experimentos.

Esta variedad de NC se recoge en dos ficheros fuente (con sus correspondientes *.h* y *.cc*). Uno de ellos, al igual que en casos anteriores, describe la cabecera del protocolo (*IntraFlowNetworkCodingHeader*). El otro fichero contiene el código necesario para lograr la comunicación entre entidades de este protocolo (*IntraFlowNetworkCodingProtocol*). A continuación se explican los diferentes métodos del módulo *IntraFlowNetworkCodingHeader*.

- **GetK, GetQ, GetNfrag, GetTx, GetSourcePort, GetDestinationPort, GetVector**: Este conjunto de métodos permite obtener cada uno de los parámetros de la cabecera correspondiente a un objeto instanciado de la clase *IntraFlowNetworkCodingHeader*. La Figura 2.7 muestra como están dispuestos cada uno de los campos en la cabecera de los paquetes de *intra-flow* NC.
- **SetK, SetQ, SetNfrag, SetTx, SetSourcePort, SetDestinationPort, SetVector**: Se trata de una colección de funciones que permiten fijar el valor en el campo correspondiente de una cabecera. Cada uno de los campos está explicado en la Sección 2.4.1.
- **Serialized**: Establece el tamaño correcto para cada uno de los campos de la cabecera. Se encarga de que todo quede alineado adecuadamente, para que el tamaño total de la cabecera no sea mayor del esperado y, por tanto, no se esté incurriendo en una pérdida de rendimiento.
- **Deserialized**: Obtiene cada uno de los elementos de la cabecera de manera ordenada, respetando el tamaño de cada campo.
- **GetSerializedSize**: Retorna el tamaño total de la cabecera en bytes. Incluye tanto la parte fija (*K, q, Tx, Nfrag, SourcePort, DestinatioPort*), como la variable (*Vector*).

Tras describir la cabecera del protocolo, el siguiente paso consiste en desgranar los métodos del módulo principal:

- **ReceiveFromUpperLayer**: Esta función es invocada cuando se envía un paquete de la capa de transporte (UDP) a la capa de red. Se intercepta mediante un *callback* para redirigirlo por este método, provocando el paso del paquete por la capa intermedia de NC. Recibe como argumento un mensaje UDP y los datos de capa 3 necesarios para dirigir el paquete al destino. La función forma un tipo de dato llamado *IntraFlowNetworkCodingBufferItem* que contiene las direcciones IP origen y destino, el paquete y los puertos UDP, y procede a almacenarlo en un buffer a la espera de disponer de suficientes para codificar. Si el paquete no es del tamaño suficiente (250 bytes) se envía directamente al nivel de red.

- **Encode**: Una vez se ha logrado disponer de K paquetes en el buffer para codificar, esta función se encarga de generar paquetes codificados a partir de los nativos. Crea la cabecera de NC, añade los datos codificados y, mediante un *callback*, envía el paquete a la capa IP. Este último paso devuelve el paquete al cauce normal de datos en una comunicación, si no se hubiera interceptado para incluir la capa de NC.

- **Receive**: Este método modela la recepción de un paquete en el destino. En primer lugar, extrae la cabecera NC y lee los datos. Si es un paquete de datos normal ($Tx = 0$), y está dentro del número de fragmento, introduce el vector en la matriz de paquetes novedosos y comprueba el rango de la matriz para determinar si el paquete aporta nueva información (si el rango es mayor). En caso afirmativo almacena el paquete en el buffer de recepción, a la espera de disponer de K paquetes para decodificar; en caso negativo descarta el paquete y elimina el vector de la matriz. Las operaciones relativas al tratamiento de vectores y matrices se han realizado mediante dos librerías, la IT++ para GF(2) y FFLAS/FFPACK para cuerpos de orden mayor GF(Q). Ambas se comentarán más adelante en este mismo documento. En el caso de que el paquete sea un ACK, el nodo actualiza el número de fragmento en caso de que sea necesario y elimina los elementos que permanecían en el buffer de transmisión, tanto del nivel NC como de 802.11. Los paquetes fuera de fragmento provocan que el nodo que los recibe envíe un ACK para hacer saber que es necesario actualizar el número de fragmento.

- **Decode**: Una vez se tienen K paquetes linealmente independientes en el buffer de recepción se llama a este método. Su cometido principal es invertir la matriz de vectores de coeficientes C y entregar, de manera simultánea, a la capa superior los K paquetes nativos originales tras decodificar con la inversa C^{-1}.

- **SendAck**: Esta función se invoca cada vez que se decodifica un fragmento. Genera un ACK y lo envía para confirmar la correcta recepción del fragmento. También se llama a este método cuando se recibe un paquete fuera de fragmento, con el fin de notificar a los nodos, que no lo hayan hecho aún, que hay que actualizar el número correspondiente.

- **ParseForwardingReception**: Se trata de la función con la que los nodos intermedios reciben un paquete. Existen tres modos de operación. El primero de ellos simplemente realiza un reenvío del paquete recibido sin comprobar si aporta información adicional. El segundo funciona de una manera similar, aunque el módulo contiene una matriz interna C donde se almacenan los paquetes novedosos (en clara analogía con el nodo destino y el método *Receive*). Si el nuevo paquete que llega es linealmente independiente, el paquete se reenvía al siguiente salto de la ruta. Por último, el tercer modo de operación es el que implementa NC. En este caso, tras comprobar si el paquete es novedoso y almacenarlo en la matriz, la función comprueba si se dispone de los T paquetes definidos al inicio. En caso afirmativo, se desencadena el procedimiento de recodificación. Cuando se reenvía un paquete, el paso del mismo al

nivel de red se realiza como en los casos anteriores, mediante un *callback* al método *IpV4L3Protocol::IpForward*. Si el modo de operación es NC, no se realiza un *callback* sino que se llama a la función *Recode*.

- **Recode**: Su cometido es el de generar un nuevo paquete y enviarlo al siguiente salto. Para ello genera un vector aleatorio de dimensión $1 \times K$, que se multiplica por la matriz de coeficientes C. Como resultado se obtiene otro vector que ocupa el lugar correspondiente en la cabecera del nuevo paquete codificado. Tras ello se envía al nivel de red para que pueda ser reenviado.

- **ReceivePromiscuous**: Este método se llama mediante un *callback* desde el nivel 802.11 cuando se recibe un paquete que no va dirigido al propio nodo. La función comprueba las direcciones MAC e IP. Una vez hecho, si determina que se puede realizar la recepción promiscua, llama a las funciones *ParseForwardingReception* o *Receive*, en función del nodo que esté ejecutando la escucha (nodo intermedio o nodo destino, respectivamente). Gracias a esta mecánica es posible adelantar información y así llenar la matriz de coeficientes más rápidamente.

- **ChangeFragment**: Su cometido es el de actualizar el número de fragmento al que tiene acceso el nodo en cuestión. Se invoca tras la recepción de un ACK con número de fragmento superior al que tiene el nodo.

- **GenerateRandomVector**: Es la herramienta utilizada para generar los vectores aleatorios, tanto en la fuente como en los nodos intermedios, durante el proceso de recodificación. Permite generar valores entre unos límites establecidos, en este caso serán 0 y $2^q - 1$.

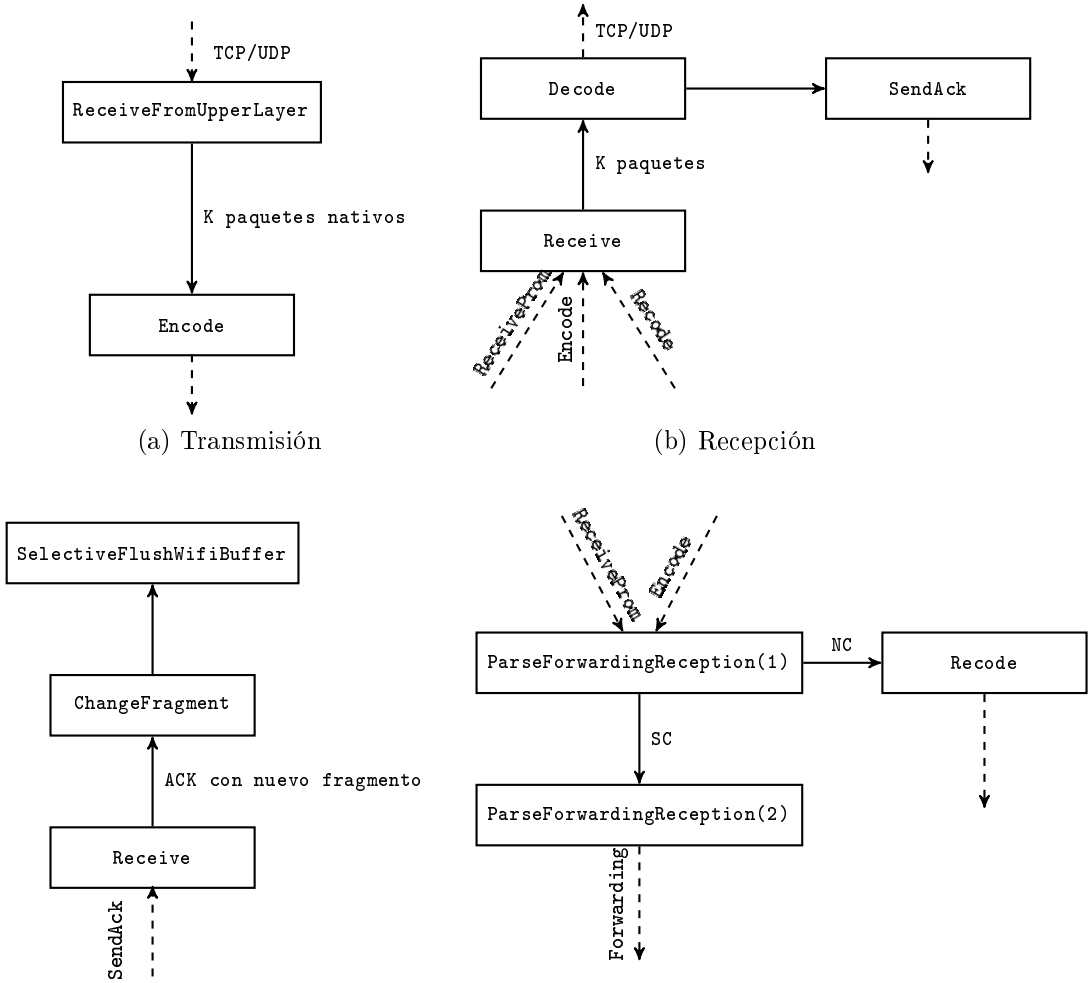
- **ResetMatrices**: Esta función realiza todo el proceso de reinicio de las matrices utilizadas cuando se actualiza el número de fragmento, comenzando la transmisión del siguiente.

- **FlushWiFiBuffer**: El objetivo de este método es el de eliminar del buffer 802.11 los paquetes que estén fuera de fragmento una vez que se recibe un ACK. Estos paquetes pueden provocar que otros que deberían tener prioridad vean retrasada su entrega y, por lo tanto, repercutir en el rendimiento. La llamada a esta función ejecuta un *callback* hacia el módulo *WiFiMacQueue*, más concretamente al método *Flush* de esta clase.

La Figura 4.6 muestra el flujo de llamadas a las funciones de este protocolo que producen los paquetes desde que se generan hasta que son confirmados.

Cabe destacar el uso de dos herramientas importantes para la correcta implementación de estos mecanismos. Ambas están relacionadas con la capacidad del código de gestionar más de un flujo de datos. La primera de ellas es el uso de variables de tipo *map*

para identificar con independencia los datos de cada uno de los flujos (variable *flowId* en el código fuente). Este tipo de dato identifica la información relativa a cada uno de los flujos con una clave (un entero, una cadena etc.) y permite operar sobre ellos sin afectar al resto. En este caso concreto se ha realizado el mapeo mediante un entero de 16 bits (u_int16_t). La forma de obtener ese valor entero es la segunda de las herramientas mencionadas, la función *Hash*. Su es, a partir de las direcciones IP origen y destino del paquete y de los puertos UDP, generar un valor entero de 16 bits. Para ello se vale del algoritmo MD5. Una vez se determina el valor y, por tanto, a qué flujo pertenece el paquete actual, se pueden realizar las operaciones pertinentes.



(a) Transmisión

(b) Recepción

(c) Recepción ACK

(d) Forwarding/Recodificación

Figura 4.6: Secuencia de llamadas a funciones en una transmisión

Más allá de los ficheros fuente descritos hasta ahora, hay tres módulos más que ayudan a cohesionar todo el código relativo a NC.

El primero de ellos es *NetworkCodingL4Protocol*, que contiene todos los métodos que pertenecen a las clases derivadas (*IntraFlowNetworkCoding* e *InterFlowNetworkCoding*), por ejemplo, los *callbacks*, miembros esenciales de las funciones. Esta clase deriva de Ipv4L4Protocol y, como se explicó anteriormente, son clases abstractas para poder ser introducidas entre las capas 3 y 4.

El segundo es un módulo *helper*, herramienta que está bastante generalizada en ns-3. Su cometido es el de facilitar el uso de los diferentes objetos a la hora de configurar una simulación. En el caso concreto de *NetworkCodingHelper*, es capaz de automatizar la creación e instalación de la pila de protocolos necesaria para poner en funcionamiento el mecanismo de NC en cada uno de los nodos del escenario.

Por último, existe un fichero de configuración que se encarga de recoger todas las opciones relacionadas con NC (número de paquetes a transmitir, elección entre *source coding* y *network coding*, los parámetros K y q, el despliegue del experimento, etc.). El nombre de este archivo es *network-coding-scenario.conf* y su contenido es procesado por el módulo *ConfigureScenario*.

Tras haber descrito los módulos relativos a ambos protocolos de NC, se va a proceder a realizar una breve introducción a algunos objetos que no tienen una relación directa con la técnica, pero que son necesarios para otras funciones a la hora de realizar las simulaciones.

Relativo a la configuración del escenario de simulación existen dos módulos a tener en cuenta:

- **test-scenario.cc**: Se trata del fichero que se ejecuta para realizar las simulaciones. Todas las llamadas a métodos y funciones se realizan de manera encadenada a partir del código contenido en este archivo. Configura algunos de los parámetros básicos de la simulación, tales como: el número de repeticiones, el establecimiento de los parámetros seleccionados o la forma de imprimir los resultados en las trazas.
- **ConfigureScenario(.h/.cc)**: Carga toda la configuración de nodos (posición, tabla de rutas, estado del canal y los enlaces etc.). Establece los protocolos de cada capa en los diferentes elementos de la red (capa de NC, nivel 802.11 etc.).

Es importante mencionar las dos librerías de C++ que se han utilizado para la manipulación de los elementos relacionados con los cuerpos finitos. La primera de ellas es *IT++* [20]. Contiene multitud de funciones matemáticas para álgebra lineal, optimización numérica, procesado de señal, comunicaciones y estadística. Para este proyecto sólo se ha utilizado su capacidad para trabajar con cuerpos finitos. Cabe destacar que dentro de este campo, *IT++* únicamente puede trabajar con cuerpos de orden 2, presentando un comportamiento óptimo para ello.

Debido a esta limitación, ha sido necesario buscar otra librería para trabajar con cuerpos de orden mayor. La librería seleccionada ha sido *FFLAS/FFPACK* [21]. Su principal característica es que permite todo tipo de operaciones de álgebra lineal con cuerpo finitos de cualquier orden (al contrario que *IT++*); sin embargo el nivel de optimización en las rutinas (tiempo de cálculo de rango e inversión) no es tan bueno como el de aquella.

4.4 Problemática

En esta sección se presentan una serie de situaciones surgidas durante el desarrollo del proyecto, que no estaban planteadas en la especificación inicial y que han propiciado la toma de distintas decisiones para abordarlas con garantía.

La primera de ellas es un fallo conocido en la librería *FFLAS/FFPACK*. A causa del mismo surgían una serie de problemas de compilación. La solución es sencilla, y consiste en declarar un *namespace* anónimo dentro del método *Failure* de la clase debug.h.

Otro problema a tratar es qué acciones tomar con un paquete que ha sido recibido mediante recepción promiscua (*IntraFlowNetworkCoding::ReceivePromiscuous*) pero que debe ser reenviado. Este caso se produce en escenarios en los que la ruta que siguen los paquetes para llegar al destino contiene una cadena de más de tres nodos.

Por último, tras realizar una serie de simulaciones para comprobar el funcionamiento de la implementación llevada a cabo, se observó un comportamiento anómalo en la recepción de los ACK en el nodo origen. Además, se transmitían un número excesivo de paquetes desfasados una vez se había actualizado el número de fragmento, ocupando de manera innecesaria el canal, e impidiendo la llegada de paquetes con información nueva o ACK's.

4.5 Soluciones adoptadas

Finalizando este capítulo, se van a plantear las soluciones tomadas, presentando además algunas opciones barajadas pero descartadas finalmente. Van especialmente dirigidas al último supuesto comentado en la Sección 4.4.

Para subsanar el problema de la recepción promiscua en los nodos intermedios, se realiza un proceso algo complejo. Cuando un paquete llega a nivel físico a un nodo intermedio se realiza un *callback* al nivel de NC, más concretamente a la función *ReceivePromiscuous*, como ya se había visto anteriormente. Este comprueba, mediante el método *AmIDestination*, si es el destinatario del paquete; en caso de no serlo (la dirección MAC y la dirección IP no coinciden con las propias) se realiza un *callback* al nivel de red para entregar el pa-

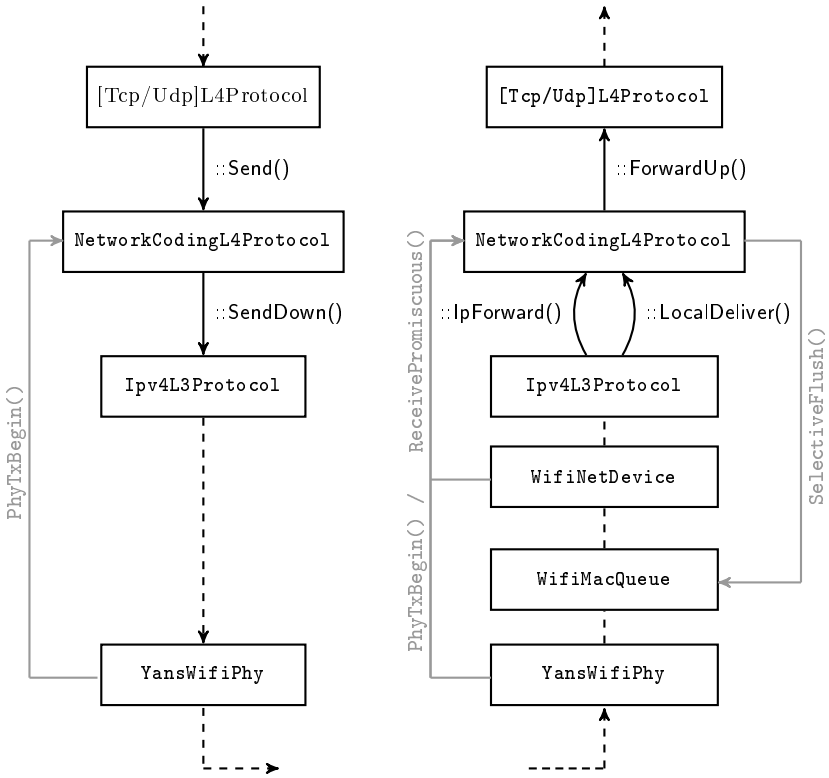
quete a ese nivel. Desde ahí sigue el curso normal y, por tanto, se entrega de nuevo al nivel de NC que lo trata con el método *ParseForwardingReception*. De este modo, un paquete que sería ignorado en un escenario normal, es procesado por si aporta nueva información.

Con el fin de evitar la saturación del canal por exceso de transmisiones, se plantearon una serie de soluciones, algunas de ellas descartadas. La primera consistía en llevar un contador de paquetes en el buffer de NC. A partir de cierto número de paquetes toda nueva recepción se descarta sin ningún tipo de distinción. Esto presenta problemas evidentes, esencialmente el hecho de que el paquete realiza todo el recorrido hasta el nivel de NC para luego ser descartado, lo que penaliza el rendimiento. Siguiendo con esta idea, se planteó la opción de borrar el paquete en el nivel físico, sin embargo, el puntero que identifica al paquete en el recorrido por los módulos 802.11 varía en cada uno de ellos (Figura 4.3). Por tanto el paquete en *WiFiMacQueue* (módulo dónde se pretende borrar el paquete) tiene una dirección de memoria diferente a cuando se recibe el paquete en *YansWiFiChannel*. A la vista de este problema la idea fue descartada. La solución final consiste en una interacción *cross-layer* entre el nivel NC y el nivel 802.11. Cada vez que se envía un paquete desde NC a los niveles inferiores se aumenta un contador. Una vez el paquete sale del buffer de nivel físico, se decrementa el contador. Los nodos, antes de generar un paquete, comprueban esta variable, si es 0 se genera un paquete nuevo. Esto se produce mediante un evento con un *callback* al llamar a la función *WiFiBufferEvent*. Gracias a esto, se tiene el canal ocupado durante todo el proceso, pero no se llega a saturar.

Aún regulando la cantidad de transmisiones que se inyectan en la red, es posible que se acumule un determinado número de paquetes desfasados, debido al retardo del canal. Para ello se diseñó la función *FlushWiFiBuffer*, que elimina los paquetes fuera de fragmento de los nodos. Sin embargo, esto conlleva el problema de que, al llamar a este método, es bastante probable que se eliminen los ACK que están esperando para ser entregados. Esto puede ser crítico, ya que elimina toda posibilidad de que el número de fragmento se actualice correctamente. Para subsanar esta tara se propuso la solución temporal de una actualización instantánea, aprovechando las posibilidades del entorno del entorno del simulador. Así, se elimina el envío del ACK, y se sustituye por una comprobación en el momento de generar un paquete. Si el número de fragmento del receptor es mayor que el del origen, se limpia el buffer, se actualiza el número de fragmento y se genera el nuevo paquete de manera adecuada. Esto es suficiente para lograr un funcionamiento correcto del protocolo, sin embargo, se prescinde de un elemento muy importante (reconocimiento de paquetes) y, por tanto, es necesario buscar una solución alternativa.

Para resolver esta situación se opta por una solución conjunta. La primera parte consiste en dotar de prioridad a los paquetes de tipo ACK en el buffer de transmisión de 802.11 de los nodos. Junto a esto se decide definir una función similar a *FlushWiFiBuffer*, pero que sea capaz de seleccionar los paquetes de un flujo de datos concreto. Esta función, llamada *SelectiveFlushWiFiBuffer*, recibe como argumento el hash (*flowId*) del flujo de datos a eliminar del buffer 802.11. Gracias a esto se evita que la recepción de un ACK de

un flujo pueda afectar a otros. La Figura 4.7 muestra el recorrido que siguen los paquetes a través de la pila de protocolos de ns-3.



(a) Transmisión (b) Recepción/Retransmisión

Figura 4.7: Flujo de los paquetes a través de la pila de protocolos de ns-3

5

Simulaciones y Resultados

En este capítulo la contribución más importante del proyecto, los resultados que ponen de manifiest si el uso de las técnicas que se han mencionado a lo largo del documento son viables y ofrecen mejoras. En primer lugar, se exponen una serie de conceptos importantes para la correcta comprensión de las simulaciones de este capítulo. Tras ello, se desglosan las condiciones con las que se configuran los experimentos cuyos resultados se presentan en la Sección 5.4. Además, se muestran y explican los escenarios que se han analizado,justificando su selección en las simulaciones. Por último, se exponen los resultados obtenidos comentándolos con el objetivo de discernir si las técnicas aplicadas cumplen los supuestos planteados.

5.1 Conceptos

Los conceptos fundamentales y métricas necesarias para comprender las simulaciones de la Sección 5.4 se presentan a continuación.

- **Throughput**: Este parámetro se define como el cociente entre la cantidad total de datos útiles transmitidos y el tiempo total que ha durado la transmisión. Tiene una gran importancia ya que aporta información sobre el rendimiento obtenido en la comunicación.

$$Throughput = \frac{\text{número de bytes de datos}}{\text{tiempo total de transmisión}} \tag{5.1}$$

- **K**: Denota la cantidad de paquetes nativos que se combinan para generar un nuevo paquete codificado. Para decodificar un fragmento también es necesario disponer de K paquetes linealmente independientes.

- **q**: Número de bits que codifican cada elemento del vector de coeficientes.

- **Esquema de recodificación**: Existen diversas formas de proceder en los nodos intermedios de la red. La primera de ellas consiste en reenviar todos los paquetes que lleguen al nodo (*naive source coding*). Como mejora inmediata surge la posibilidad de dotar de inteligencia a los nodos intermedios, y que sean capaces de distinguir los paquetes que son combinación lineal de aquellos recibidos previamente; si así fuera, el paquete se descartaría, en caso contrario se reenviaría de manera normal (*smart source coding*). Por último, se analiza también el esquema de codificación de red ya comentado previamente. En él, los nodos mantienen una matriz con paquetes codificados y, cada vez que llega uno nuevo al nodo, se genera un paquete recodificado a partir de los que ya dispone en el buffer (*network coding*).

- **Modo de transmisión al canal**: El tercer bloque de simulaciones se presenta como un escenario donde varios nodos intermedios transmiten al nodo destino. En casos como este, es necesario determinar como se van a inyectar los paquetes en el canal. En una primera aproximación, todos los paquetes que se generan son enviados al nivel físico. Esto puede provocar una saturación del canal y, por tanto, efectos adversos en el rendimiento de la transmisión. Para solucionar esto, se plantea un modo de transmisión probabilístico en el que los nodos manden los paquetes a los niveles inferiores con una probabilidad $1/N$, siendo N el número de nodos intermedios que transmiten al nodo destino. De esta forma, se pretende garantizar que un paquete siempre sea retransmitido, pero que el canal no se sature, puesto que el acceso al nivel físico se divide de manera equiprobable. Sin embargo, este método plantea un problema fundamental, y es que se está asignando la misma prioridad a nodos cuyos enlaces con el destino podrían tener una FER alta que a otros con una mejor calidad en el canal. Para ello se plantea una solución *cross-layer*. Mediante una monitorización del nivel físico se puede determinar la FER de cada uno de los enlaces. A partir de la misma, el nivel de NC determina la probabilidad de que el paquete se envíe a los niveles inferiores. La expresión de la probabilidad con la que se procederá al envío del paquete se puede observar en la Ecuación 5.2, siendo k una constante dada por la Ecuación 5.3. N representa, al igual que antes, el número de nodos intermedios que están intentando transmitir hacia el destino.

$$Pr\{i\} = \frac{k}{FER_i} \tag{5.2}$$

$$k = \frac{1}{\sum_{j=0}^{N-1} (FER)^{-1}} \tag{5.3}$$

- **Retardo**: Este parámetro permite analizar el tiempo que tarda en completarse la recepción de un fragmento respecto al anterior. Es importante, puesto que está íntimamente ligado al *throughput* de una transmisión. Despliegues con un nivel de complejidad elevado podrían derivar en un aumento considerable de este parámetro, acorde al número de saltos de la red.

5.2 Proceso de medida

En esta sección se detallan los diferentes parámetros que se han establecido para llevar a cabo las simulaciones.

- El número de ejecuciones de cada uno de los escenarios varía en función de la naturaleza del experimento. Se ha podido comprobar que, en general, la varianza de los resultados en estos escenarios no es excesivamente grande, por lo que en algún caso, el número de ejecuciones se ha reducido para aligerar el proceso. Para la primera tanda de simulaciones, se han realizado 100 experimentos por cada análisis (par de valores K y q). En el segundo bloque se han realizado 20 por cada valor de FER. Por último, para la tercera tanda, se ha utilizado el método de Monte Carlo para asegurar la validez estadística de los resultados. Se han realizado 200 simulaciones de cada uno de los casos posibles para, poder analizar las correspondientes cdf. Las medidas aparecen representadas mediante el valor medio, junto con el intervalo de confianza del 95 %, que se ha calculado con la expresión de la Ecuación 5.4

$$\text{Intervalo de Confianza} = 2.2281 \cdot \frac{\sqrt{\text{varianza medidas}}}{\sqrt{\text{número de medidas}}} \tag{5.4}$$

- Durante los experimentos se ha trabajado con una *Maximum Transfer Unit* (MTU) de 1500 bytes, con lo que el tamaño de paquete generado a nivel de aplicación es el que muestra la Ecuación 5.5.

$$MTU - UDP_{header} - IP_{header} = 1500Bytes - 8Bytes - 20Bytes = 1472Bytes \tag{5.5}$$

- La tecnología utilizada para el nivel físico y de acceso al medio ha sido IEEE 802.11b. Este estándar establece velocidades máximas de transmisión de 11 *Mbps* y mínimas de 1 *Mbps*. En este proyecto se ha operado siempre con el valor máximo.

- Otro parámetro muy interesante es el número de transmisiones 802.11 que pueden realizarse en caso de pérdida de una trama. Un valor de 1 implicaría que las retransmisiones están deshabilitadas. Un número habitual en dispositivos reales es 4. A causa del propio mecanismo de retransmisiones de 802.11, es posible que el rendimiento de NC se vea perjudicado, ya que, si se pierde un paquete, inmediatamente es reemplazado por otro que contiene la misma cantidad de información, sin necesidad del proceso que efectúa 802.11 para recuperar la trama. Sin embargo, en el caso del protocolo TCP activo, las retransmisiones a nivel físico pueden ayudar a que no sea necesario que entre en juego el propio sistema de recuperación de errores del protocolo TCP que, como se ha comentado anteriormente, incurre en una penalización importante para el rendimiento. Para los experimentos en los que no se especifique, se ha establecido de forma predeterminada 4 para TCP y 1 para NC.

Estas características se van a utilizar en todos los experimentos de forma general salvo que se especifique lo contrario en la Sección 5.4. El resto de parámetros variarán para realizar un estudio en profundidad de su impacto sobre el rendimiento.

5.3 Escenarios

Los escenarios utilizados en las simulaciones han sido tres. A continuación, se exponen sus características principales:

5.3.1. Escenario de dos nodos

Es el más simple de los que se van a analizar a lo largo de este capítulo. La topología se puede observar en la Figura 5.1; consiste en dos nodos que están dentro del rango de comunicación del otro y en el que uno de ellos ejerce como nodo fuente (S) y el otro como destino (D). El enlace que los une está caracterizado por una FER que puede variar a lo largo del experimento, con el fin de caracterizar el comportamiento en distintas situaciones.

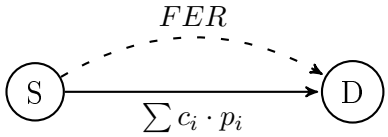


Figura 5.1: Escenario de dos nodos

El nodo origen genera datos a nivel de aplicación, que se combinan dando lugar a paquetes codificados en la capa de RLC, que se envían al destino. Este nodo, como se ha explicado anteriormente, los almacena hasta que dispone de los suficientes para ser capaz de decodificarlos.

Este escenario permite acometer un primer análisis de cómo influyen los parámetros básicos en el rendimiento de la transmisión. También permite estudiar el impacto de las retransmisiones 802.11 cuando las condiciones del canal empeoran de forma progresiva. Por tanto, se modificarán los valores de K y q, con el fin de obtener una configuración óptima para usar en las posteriores simulaciones.

5.3.2. Escenario de tres nodos

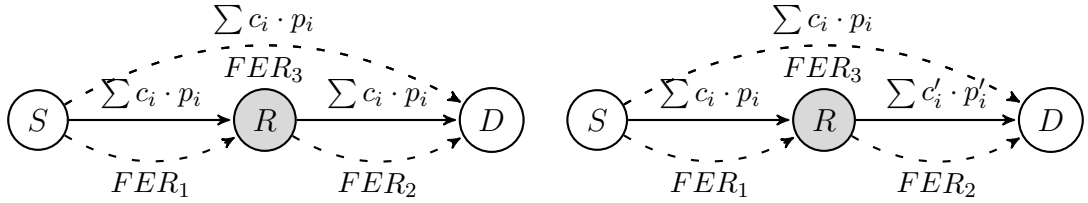
En este escenario se despliegan tres nodos. El nodo fuente (S) genera paquetes de una manera análoga a como ocurre en el caso anterior. El nodo intermedio (R) se encarga

de recibir y reenviar los paquetes que vayan dirigidos a otros nodos. Por su parte, el nodo destino (D), al igual que en el caso anterior, almacena y decodifica los datos contenidos en los paquetes.

En este despliegue se habilita la posibilidad de que los nodos extremos realicen *overhearing*. Los enlaces vienen caracterizados por distintos valores variables de FER, independientes entre ellos, permitiendo que se puedan configurar por separado para establecer el escenario deseado.

Se pretende analizar el impacto que tienen los distintos paradigmas de transmisión, véase el clásico método de *store-and-forward* o el objeto de estudio de este proyecto, NC, sobre el rendimiento de la transmisión. Tras las pruebas realizadas en el primer escenario (ver Sección 5.4.1), se establece el uso de los parámetros $K = 64$ y $GF(2)$.

La Figura 5.2 muestra los dos escenarios analizados. En el primero (*Random Linear Source Coding* (RLSC)) el nodo intermedio aplica *store-and-forward*, mientras que el segundo (RLNC) añade la posibilidad de llevar a cabo NC en el nodo intermedio.



(a) RLSC (b) RLNC

Figura 5.2: Escenarios de tres nodos

5.3.3. Escenario multi-nodo

Por último se plantea un escenario en el que existen varios nodos intermedios capaces de retransmitir paquetes al nodo destino. Estos enlaces podrían representar diferentes rutas en un posible escenario multi-camino, como es habitual en WMN. Cada uno de los enlaces desde los nodos intermedios al destino se caracteriza con una FER aleatoria, distinta a los demás. Por el contrario, la FER de los enlaces del origen a los nodos intermedios se mantiene fija a 0. En la primera tanda de simulaciones se deshabilitará la opción de *overhearing*. En un segundo bloque se activará, fijando la FER del enlace directo entre origen y destino a 0.6.

El objetivo principal de este escenario es comprobar que las características multi-camino de las WMN pueden contribuir a lograr mejoras en el rendimiento. Además se estudiará si es adecuado regular el número de transmisiones que puede realizar cada nodo

en función del estado del canal. Se analizará para escenarios que disponen de 2 a 8 nodos intermedios. La configuración, como en el apartado anterior, se establece para valores de $K = 64$ y $GF(2)$.

La Figura 5.3 muestra la disposición de los nodos en este despliegue.

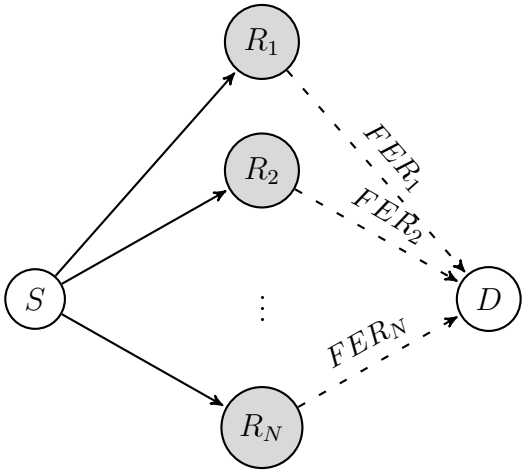


Figura 5.3: Escenario multi-nodo

5.4 Resultados

En esta sección se muestran los resultados obtenidos durante la simulación de los escenarios descritos en la Sección 5.3. Posteriormente, se analizan con el fin de determinar si son satisfactorios en cuanto a las mejoras que aportan frente a las técnicas utilizadas tradicionalmente, describiendo las conclusiones pertinentes.

5.4.1. Escenario de dos nodos

El cometido principal de este escenario es realizar un estudio de la influencia de los parámetros característicos del RLC sobre la transmisión de una cantidad determinada de datos. Se elegirán los valores que optimicen el rendimiento, tanto en *throughput* como en el procesado de datos en los nodos. Como se ha comentado anteriormente, se han realizado 100 experimentos independientes por cada par de valores (barrido en K y Q), consistentes en el envío de un fichero de 15MB (10000 paquetes aproximadamente) desde el nodo origen al nodo destino. El canal se supone ideal, lo que implica que todos los paquetes que se transmitan en exceso serán resultado de las combinaciones lineales en el momento de generar el vector de codificación.

En primer lugar, la Figura 5.4 muestra el *throughput* máximo alcanzado en las simulaciones. A la vista de la gráfica se puede afirmar que el rendimiento está comprendido entre 2.6 *Mbps* y 5.7 *Mbps*.

La Figura 5.5 muestra como evoluciona el rendimiento máximo del canal a medida que aumenta la carga en él, observando un efecto de saturación. Cabe destacar que los valores máximos alcanzables para este tipo de canal con los protocolos TCP y UDP son, respectivamente, ~ 5 *Mbps* y ~ 6 *Mbps*. Por tanto, se puede afirmar que el uso técnicas de RLC combinadas con UDP y unos parámetros adecuados, puede alcanzar unos valores de *throughput* muy cercanos al rendimiento del UDP tradicional. Sin embargo, también aporta las funcionalidades comentadas en el Capítulo 2, lo que supone una ventaja frente a ambos protocolos de transporte.

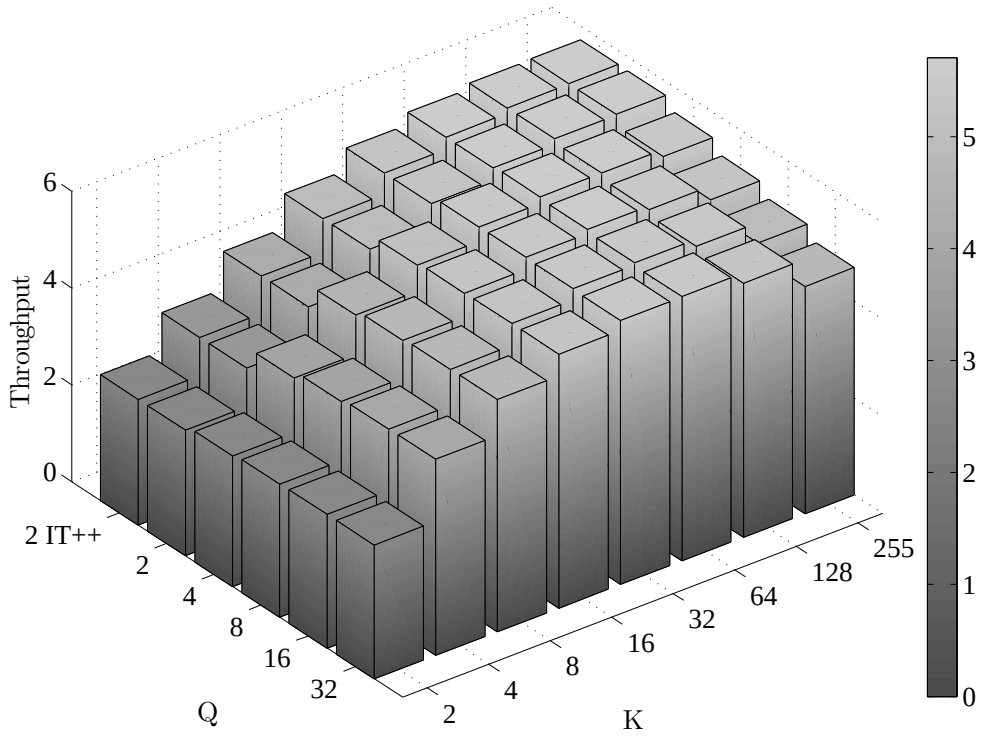


Figura 5.4: Throughput para cada valor de K y Q

La Figura 5.6 presenta los resultados para facilitar un estudio en profundidad de la influencia de los parámetros K y Q. Se puede observar que, para valores pequeños de K, el *throughput* se mantiene a niveles bastante bajos. La causa es la escasa cantidad de posibles vectores de coeficientes a la hora de codificar un paquete de un fragmento; por tanto, la cantidad de paquetes linealmente dependientes es muy grande. A medida que K crece, este número disminuye, estabilizándose a partir de los valores $K = 32$ y $K = 64$. Por otro lado, mientras aumenta la longitud del vector de coeficientes, la posibilidad de que los

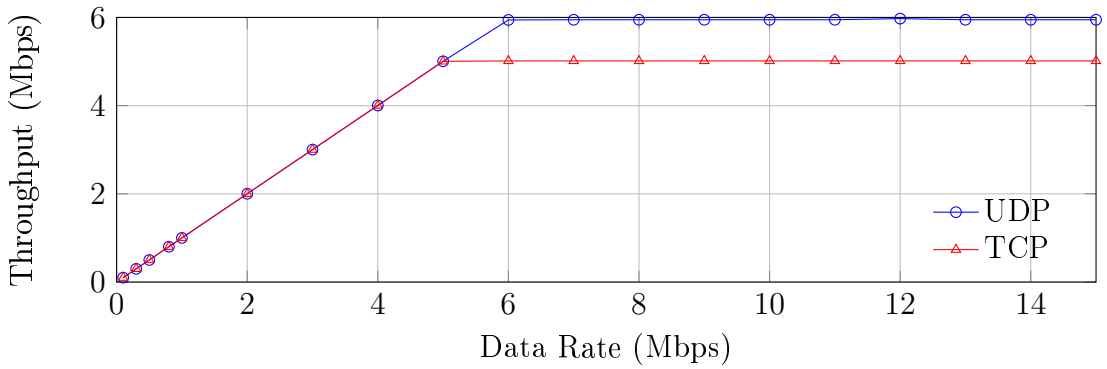


Figura 5.5: Rendimiento máximo en el canal

vectores sean linealmente independientes también crece. Una vez se alcanzan los puntos mencionados, se puede apreciar que la cantidad de paquetes descartados por dependencia lineal es muy reducida como se verá más adelante.

Por otro lado, el parámetro Q, que indica el número de posibilidades existentes para cada uno de los coeficientes del vector, influye de una manera similar. A medida que crece, la cantidad de combinaciones lineales disminuye. Esto se puede observar en las curvas correspondientes a Q altos; cuando $K < 32$, se puede observar que están por encima del resto de curvas.

Sin embargo, para valores de K altos, el comportamiento de estas configuraciones se invierte, quedando $Q = 2$ por encima del resto. La razón es que, al igual que con K, a partir de cierto punto, aumentar este valor no derivará en una mejora del porcentaje de paquetes linealmente dependientes recibidos. Al contrario, incluir el vector de coeficientes con valores elevados de K y Q implica que la cantidad de datos que transporta cada uno de los paquetes es menor y, por tanto, el *throughput* se ve perjudicado.

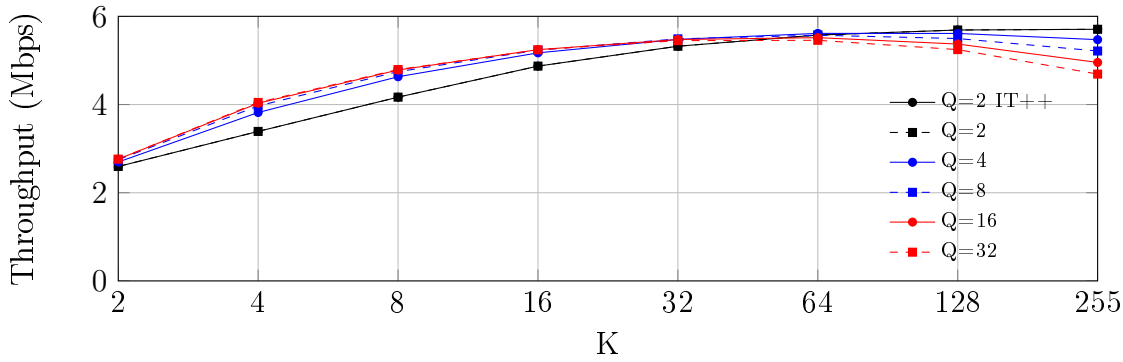


Figura 5.6: Throughput en un canal sin pérdidas

En conclusión, los valores bajos en ambos parámetros ponen de manifiesto el perjuicio que provoca tanto el descarte de paquetes linealmente dependientes como el envío de

ACK's tras cada fragmento. Como se verá más adelante, la penalización por ACK no es despreciable e, incluso, en cierto casos, puede ser crítica.

La Figura 5.7 muestra el porcentaje de paquetes en exceso que se envían a causa de los descartes en el receptor. Se puede observar el efecto comentado con anterioridad. Valores bajos de ambos parámetros provocan un aumento drástico del porcentaje de paquetes descartados. Con $Q = 2$ se alcanza hasta un 30 % de transmisiones innecesarias, lo cual es difícilmente asumible en un escenario real. A medida que Q crece, se observa un sustancial descenso en el ratio de paquetes en exceso, alrededor del doble entre los dos peores casos ($Q = 2$ y $Q = 4$).

Por otro lado, al aumentar el valor de K, dicho porcentaje cae drásticamente, obteniendo valores de 2 % para $K = 64$ y $Q = 2$. Esta cifra se acerca más a lo buscado para lograr una transmisión eficiente. A partir de este punto, cualquier aumento en ambos parámetros repercutirá en una mejora de la cantidad de paquetes descartados.

A la vista de lo comentado anteriormente, en este escenario concreto, la relación entre el *throughput* y la cantidad de paquetes descartados es evidente, y condiciona la elección de los valores para conseguir un rendimiento adecuado.

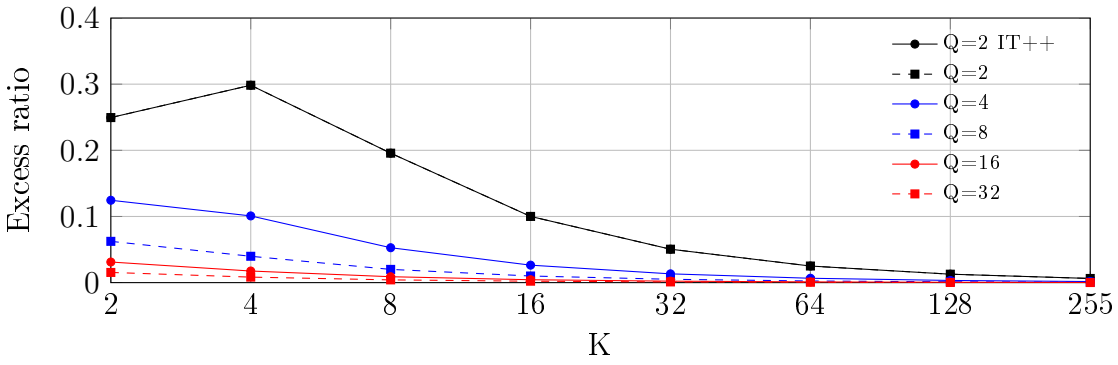


Figura 5.7: Ratio de paquetes en exceso

Como se pudo apreciar en la Sección 3.1.1, el exceso de paquetes tiene un efecto sobre el tiempo total de transmisión. El parámetro ϵ modela el porcentaje de tiempo que se ha invertido en la transmisión de paquetes innecesarios. Al contrario que el ratio de exceso, este valor se calcula sobre el tiempo total de transmisión, y no sobre el tiempo útil. En la Figura 5.8 se puede observar como evoluciona para las diferentes configuraciones.

A la vista de la gráfica, se puede observar que el factor de penalización va desde 20 % (para los peores casos) hasta 0.01 % (para casos óptimos). Sin embargo, cabe destacar que este parámetro sólo estudia la influencia del descarte de paquetes y, por tanto, no debería utilizarse en solitario para condicionar la elección de los valores de los parámetros K y Q.

A pesar de lo que pueda parecer, la transmisión de los ACK para confirmar cada fragmento puede tener consecuencias importantes en el rendimiento de la transmisión.

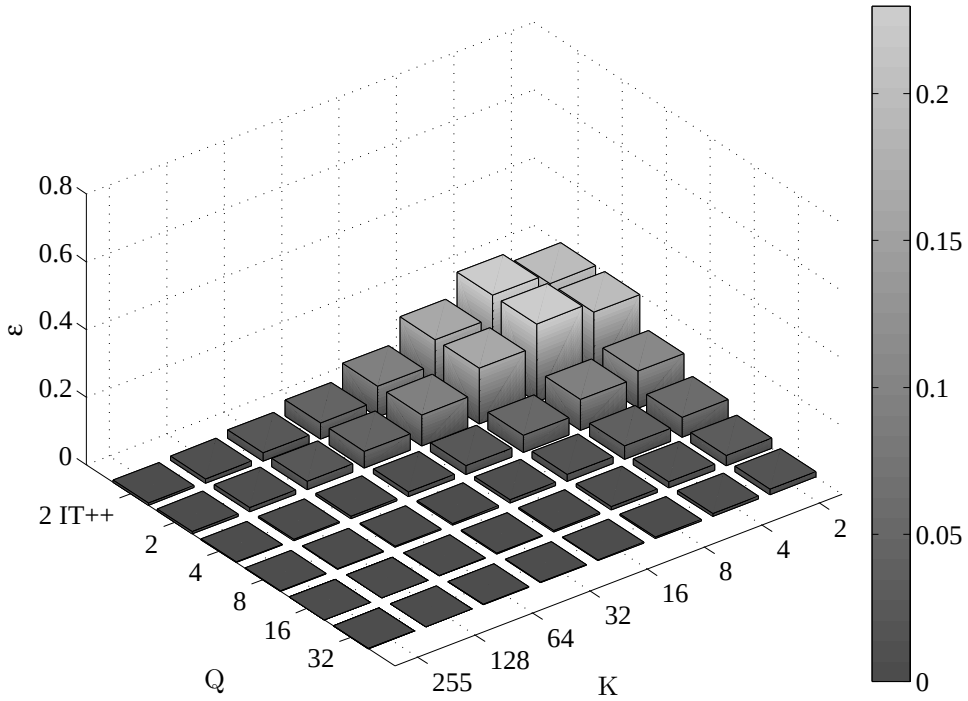


Figura 5.8: Penalización por paquetes en exceso

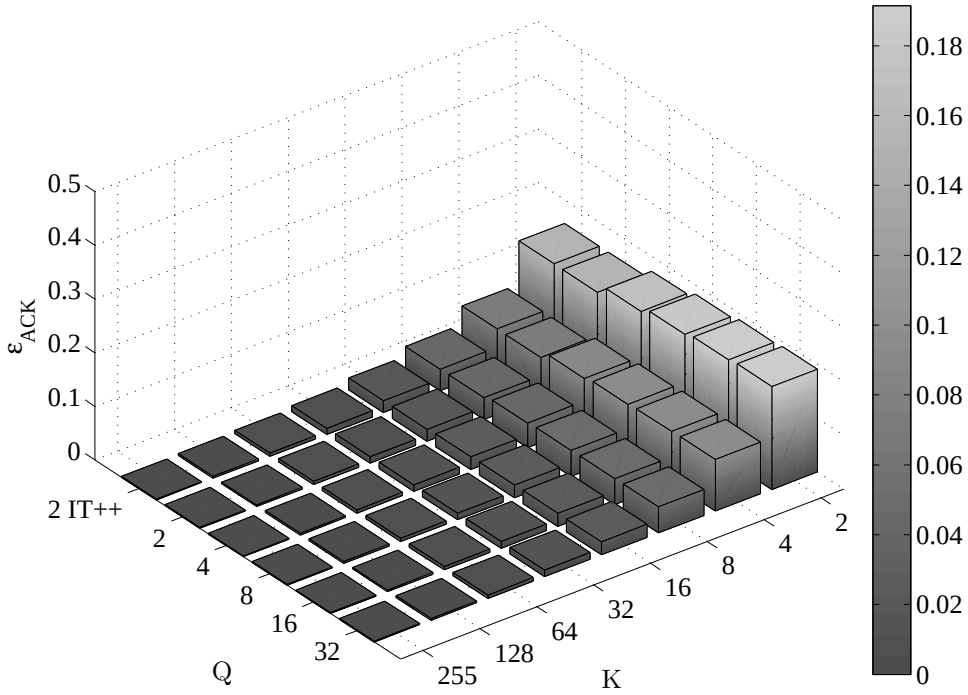


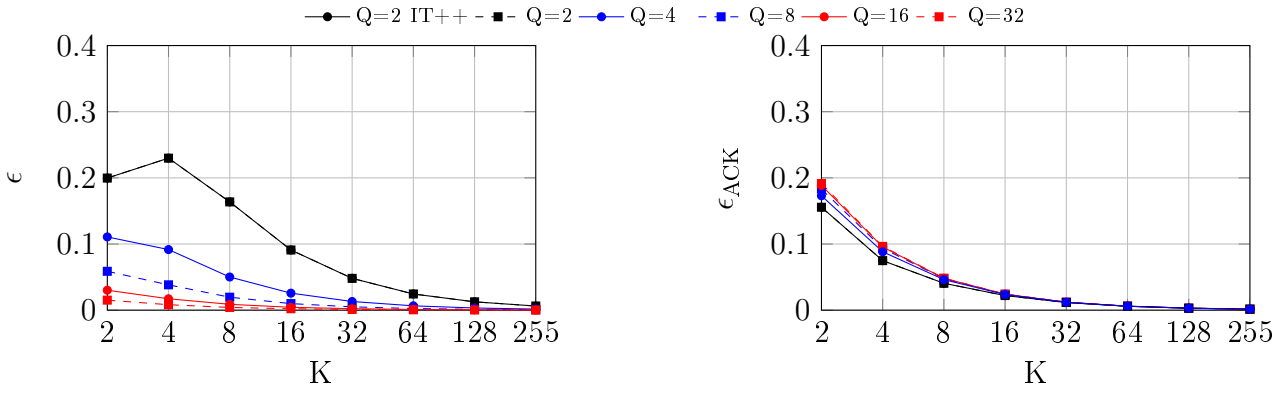
Figura 5.9: Penalización por el envío de ACK's

En la Figura 5.9 se observa su impacto. El factor ϵ_{ACK} representa el tiempo invertido en la transmisión de los ACK respecto al total del envío. El rango de valores en el que oscila va desde 20 % hasta 0.15 %. Lógicamente, una penalización del 20 % impacta muy negativamente en la transmisión.

La Figura 5.10 permite analizar ambos parámetros de manera más profunda. En primer lugar, se aprecia una clara semejanza entre la forma de la curva de ϵ y el ratio de exceso de paquetes. Esto es debido a que cada paquete en exceso tiene asociado un retardo temporal, y la combinación de ambos queda representada por el parámetro ϵ. Por tanto, la evolución con los parámetros K y Q son análogas en ambas gráficas.

Por otro lado, las curvas relativas a ϵ_{ACK} evolucionan de manera distinta. El impacto del ACK decae rápidamente con el aumento de K. Un resultado obvio si se tiene en cuenta que las confirmaciones se alternan entre K paquetes de datos (longitud de un fragmento). Así, a medida que el tamaño del fragmento aumenta, la proporción de ACK's disminuye.

También se puede observar que el parámetro Q apenas tiene relevancia sobre ϵ_{ACK}, pues la cantidad de paquetes de cada fragmento no depende explícitamente de él. Sin embargo, se aprecia una pequeña diferencia para K bajos, en los que la curva de penalización está por encima para los Q altos. Para encontrar la razón de esto hay que acudir a la Ecuación 3.5, en la que se observa que existe una relación inversa entre ϵ_{ACK} y ϵ. Puesto que ϵ sí depende de Q, se produce la pequeña variación entre curvas, que se observa en la Figura 5.10b



(a) Factor ϵ

(b) Factor ϵ_{ACK}

Figura 5.10: Factores de penalización

Una vez se ha estudiado la influencia de las diversas penalizaciones en el rendimiento, y se han seleccionado los valores de K y Q óptimos, se procede a abordar el estudio de otro aspecto importante.

Como se ha comentado anteriormente, el uso de las técnicas de RLC y NC requieren un aumento de la capacidad de almacenamiento y procesado de datos de los nodos que

intervienen en la transmisión. Aún así, siguen siendo dispositivos que tienden a estar limitados en este sentido. Por tanto, es necesario realizar un estudio para tratar de optimizar la sobrecarga relacionada con el cómputo de datos y no castigar en exceso a los nodos, pues podría influir negativamente en el rendimiento de la transmisión.

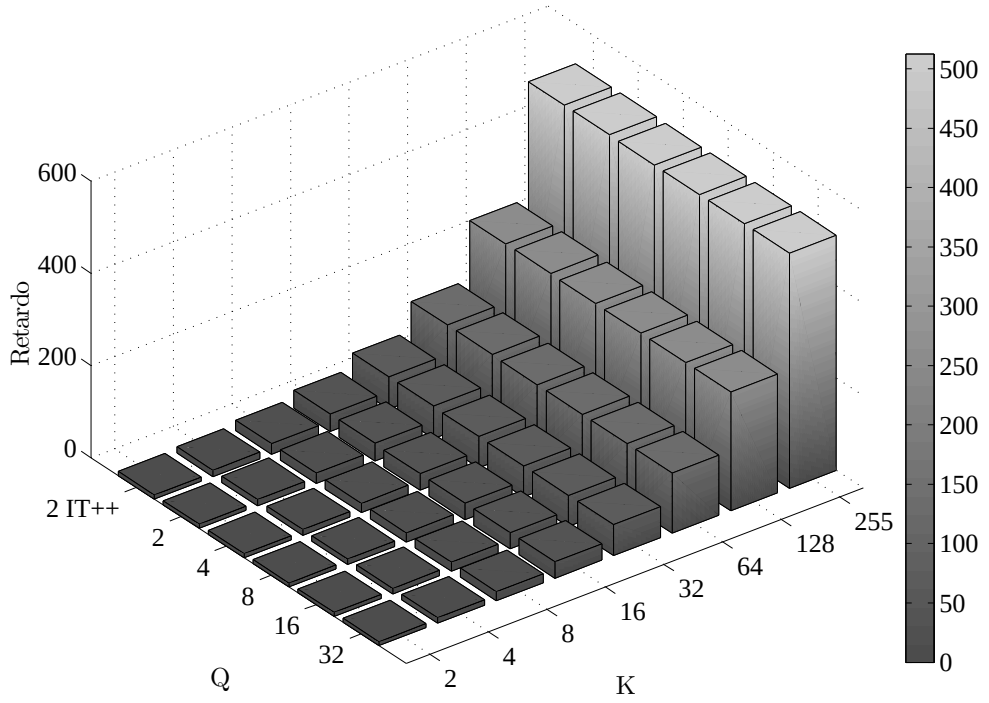


Figura 5.11: Retardo entre fragmentos consecutivos

La Figura 5.11 muestra el retardo medio que existe entra la recepción de dos fragmentos consecutivos. Lógicamente a medida que K crece, este valor aumenta. En la gráfica se aprecia que el crecimiento es exponencial. Por otro lado, no se aprecia ninguna variación en el valor de retardo al modificar Q. Esto se debe a que no influye en la cantidad de paquetes que se transmiten en cada fragmento y, por tanto, no se incurre en un mayor retardo.

La importancia de este parámetro radica en que, en ningún caso, el valor del retardo entre dos paquetes puede ser menor que el tiempo de cálculo del rango de la matriz cuando se recibe un nuevo paquete. Si así fuera, se estaría generando un "cuello de botella" que perjudicaría el rendimiento.

Por ello, las Figuras 5.12 y 5.13 muestran el tiempo de cálculo del rango y de la matriz, inversa, respectivamente para cada par de valores K, Q. En el caso de $GF(2)$ se muestran resultados con las dos librerías utilizadas en el proyecto, *IT++* y *FFLAS/FFPACK*.

A la vista de las gráficas, resulta sencillo extraer varias conclusiones. En primer lugar se aprecia que *IT++*, al estar desarrollada expresamente para trabajar con cuerpos de orden

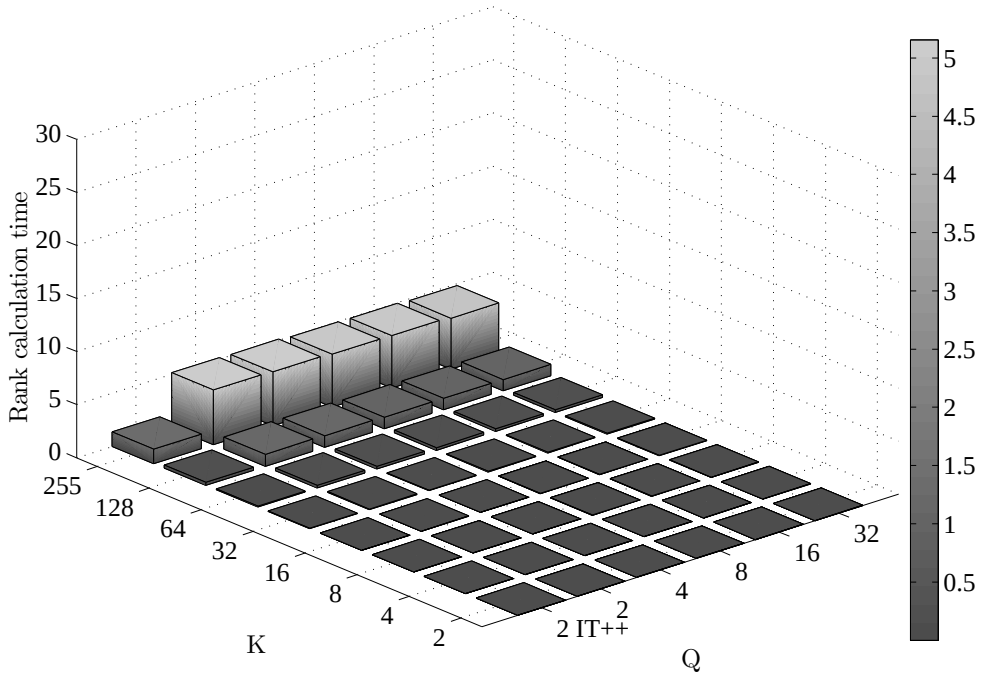


Figura 5.12: Tiempo de cálculo del rango

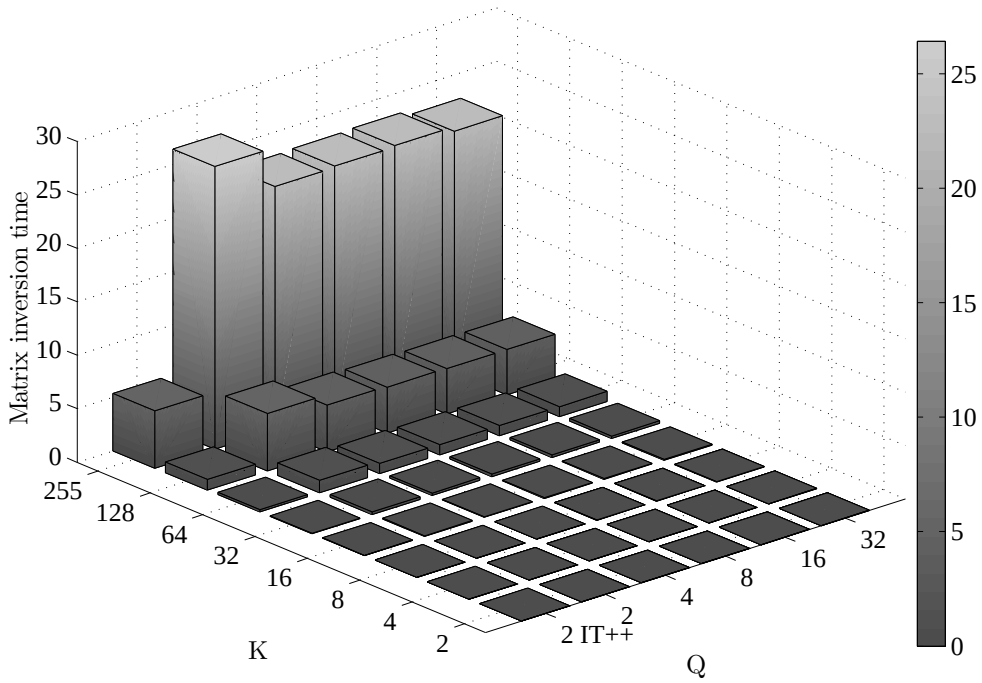


Figura 5.13: Tiempo de inversión de la matriz

2, resulta mucho más eficiente que *FFLAS/FFPACK*. En concreto, *IT++* tarda 1.3753 ms en calcular el rango para $K = 255$, frente a los 5.1569 ms que tarda *FFLAS/FFPACK*. En el caso del tiempo necesario para calcular la inversa ocurre algo similar, siendo cinco veces más rápida *IT++*, 5.3591 ms frente a 26.4387 ms. Sin embargo, si se pretende trabajar con cuerpos de orden mayor no queda otra alternativa que acudir a *FFLAS/FFPACK*.

Por otro lado, se aprecia que a medida que aumenta el tamaño de la matriz (valores de K más altos), el tiempo necesario para realizar ambas operaciones crece considerablemente. Así, los valores con $K = 128$ y $K = 255$ impiden su uso en campañas de simulación muy extensas. Además, como se comenta al principio de la sección, a partir de $K = 64$ se puede afirmar que aumentar el número de paquetes por fragmento no aporta una mejora sustancial.

Por último cabe destacar que el valor de Q no influye de manera clara en el rendimiento de las librerías, salvo en el caso de $Q = 2$ y *FFLAS/FFPACK*, configuración en la que presenta un comportamiento sorprendentemente malo.

A la vista de los resultados obtenidos hasta el momento, se plantea el uso de un valor de K lo suficientemente elevado para que no se produzca una recepción excesiva de paquetes linealmente dependientes pero que, a su vez, no perjudique notablemente la cantidad de datos transportados en cada paquete. En cuanto a Q, se ha comprobado que no tiene una influencia relevante cuando los valores de K son altos. Por tanto, los términos para la toma de decisión del parámetro se basan principalmente en el rendimiento temporal de las operaciones sobre matrices, lo que inclina la balanza hacia el uso de la librería $IT++$.

Una vez comentado esto, se establece, para futuras simulaciones, el uso de los valores $K = 64$ y $Q = 2$.

Otro aspecto importante, que merece la pena caracterizar, es la influencia de las retransmisiones 802.11 sobre el rendimiento de la transmisión. Este mecanismo se encarga de retransmitir tramas a nivel MAC en caso de existir pérdidas. En dispositivos reales, se puede configurar el número de retransmisiones posibles antes de descartar la trama. Aunque el valor más habitual es 3, se pueden encontrar casos en los que el número de retransmisiones van desde 0 a 6. Para realizar el análisis, se han establecido los parámetros $K = 64$ y $Q = 2$ y se ha configurado el canal variando la FER en el rango $[0 : 0.1 : 0.8]$.

En la Figura 5.14 se pueden apreciar las diferencias entre las diferentes configuraciones. Cuando la $FER = 0$, las tres curvas correspondientes a RLC coinciden en el mismo punto, ya que al no haber errores en el canal, el mecanismo de retransmisión 802.11 no entra en juego. TCP, como se ha visto anteriormente, es la alternativa con un rendimiento más bajo. A medida que el canal se deteriora, el throughput de las curvas $RLC(6)$ y $RLC(3)$ es menor que el observado para la alternativa sin retransmisiones. Se puede decir, por tanto, que el mecanismo de repetición de tramas 802.11 es más lento que el simple hecho de descartar el paquete, "sustituyéndolo" por uno nuevo con la misma información.

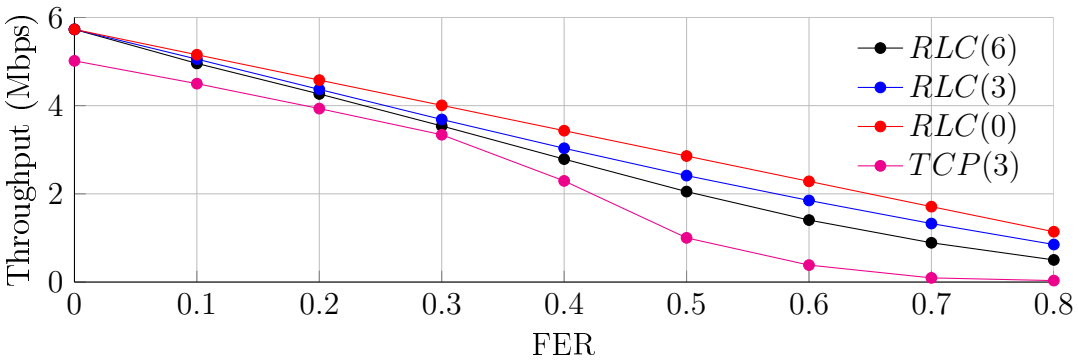


Figura 5.14: Influencia de las retransmisiones 802.11 en un canal con pérdidas

Gracias a esto se constata la utilidad de la técnica RLC como método de recuperación de información perdida. La penalización que introducen las retransmisiones 802.11 se puede apreciar, por ejemplo, para una $FER = 0.6$, en la que $RLC(0)$ alcanza un throughput de 2.3 $Mbps$, quedándose el $RLC(6)$ en 1.5 $Mbps$, lo cual implica una diferencia sustancial.

Por otro lado, se puede observar que el rendimiento de TCP empeora de manera inaceptable a medida que el canal se deteriora ($FER > 0.3$). Esto no hace más que constatar el comportamiento de TCP sobre medios hostiles (inalámbricos) se ve muy perjudicado.

A la vista de los resultados observados, se decide proseguir con las simulaciones, deshabilitando las retransmisiones 802.11.

5.4.2. Escenario de tres nodos

En esta sección se estudia el comportamiento del esquema RLC en combinación con técnicas de NC en los nodos intermedios. La topología con la que se va a trabajar se muestra en la Figura 5.2. En primer lugar, se analiza la influencia del *overhearing* sobre la transmisión. Para ello, se plantea un escenario donde los enlaces directos $S \rightarrow R$, y $R \rightarrow D$ son ideales ($FER = 0$), por lo que no habrá pérdida de paquetes derivados del canal. Por otro lado, la FER del enlace entre $S \rightarrow D$ se varía a lo largo del experimento en el rango de valores de [0:0.1:1]. Se estudiará también la influencia del tamaño del fragmento.

Los atributos de la simulación son similares al escenario anterior. Se transfiere un fichero de $15MB$, manteniendo la MTU de 1500 Bytes y, en el caso de NC, se han deshabilitado las retransmisiones 802.11. Se han realizado 50 simulaciones independientes por cada par de valores.

La Figura 5.15 muestra los resultados para el escenario mencionado, con el comportamiento *store-and-forward*, en el que el nodo intermedio solamente retransmite los paquetes que le llegan de S.

Se puede observar que el *throughput* se mantiene más o menos constante al variar la FER. A pesar de que un paquete no llegue por el salto directo de $S \rightarrow D$, la ruta alternativa (la predeterminada) es ideal y todos los paquetes alcanzan el destino. Además, la información que aporta un paquete que se ha recibido por *overhearing* no proporciona información novedosa frente a lo que va a llegar a través del nodo intermedio.

En cuanto a la evolución del *throughput* con el parámetro K, se puede apreciar que mantiene el comportamiento observado en el escenario de dos nodos. La única diferencia es que, al aumentar la distancia entre origen y destino (tanto en distancia como en número de saltos), el valor del throughput se reduce.

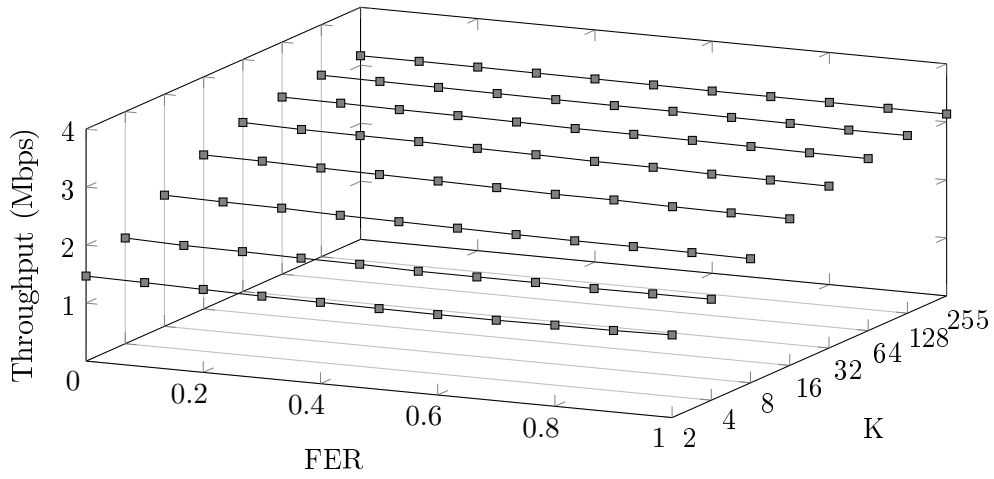


Figura 5.15: Impacto del overhearing en store-and-forward (RLSC)

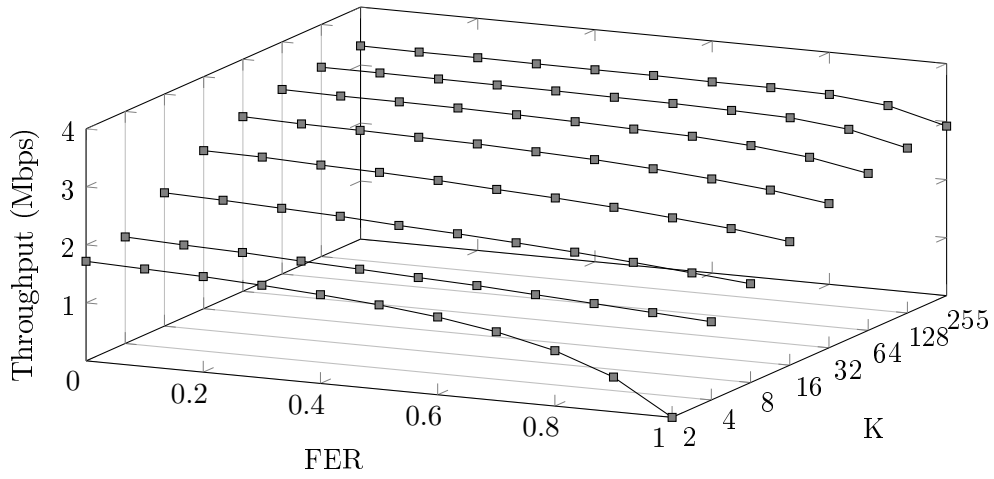


Figura 5.16: Impacto del overhearing en network coding (RLNC)

A raíz de esto, se puede concluir que el overhearing de paquetes no implica una mejora sustancial en este tipo de escenario al utilizar el comportamiento *store-and-forward* en el nodo intermedio.

A continuación se modifica el comportamiento del nodo intermedio R. En este caso, incorpora funciones de NC. Así, la información que llega al destino pertenece a dos fuentes distintas. La primera de ellas son los paquetes recibidos directamente desde el origen por medio del *overhearing*, viéndose reducidos a medida que la FER del enlace directo crece. La segunda fuente son los paquetes que genera el nodo intermedio mediante la codificación de los recibidos desde el origen.

La Figura 5.16 muestra los resultados obtenidos en las simulaciones. En primer lugar se puede observar que no existen grandes diferencias entre el uso de *store-and-forward* y NC. En este tipo de escenario es lógico, pues los paquetes que recodifica el nodo intermedio están basados en los originales y, puesto que el enlace $R \rightarrow D$ es ideal, los paquetes que provienen del *overhearing* no aportan información adicional.

A pesar de esto, se puede apreciar una ligera mejoría de RLNC. El motivo es el hecho de dotar al nodo intermedio de la inteligencia para descartar paquetes linealmente dependientes cuando los recibe de la fuente, con lo que se evita transmitir información innecesaria en el medio inalámbrico. Sin embargo, a partir de $FER > 0.6$ el deterioro del *throughput* es más pronunciado. La causa es que, al igual que en la fuente, la recodificación en los nodos intermedios es aleatoria, por lo que se da la posibilidad de que los paquetes que se generan sean linealmente dependientes, y tengan que ser descartados.

En resumen, para valores de FER bajos, RLNC aporta una ligera mejora por la capacidad de los nodos intermedios de descartar paquetes linealmente dependientes. En cambio, para valores de FER elevados, las combinaciones lineales producidas por R, junto a la reducción de la información que aporta el salto directo $S \rightarrow D$, hace que el throughput se vea perjudicado.

El siguiente paso es el estudio de la influencia de las condiciones del enlace $R \rightarrow D$ en el rendimiento de la transmisión. Con este fin, se plantea un escenario en el que la FER del enlace $S \rightarrow R$ se mantiene fija a 0 y la de $S \rightarrow D$ a 0.6. En la simulación, se realiza un barrido de la FER del enlace $R \rightarrow D$ en el rango [0:0.1:0.6]. Se espera que la información perdida en este salto pueda ser, en parte, compensada por los datos aportados por el salto directo de origen a destino mediante el *overhearing* de paquetes. En la Figura 5.17 se muestra el resultado del experimento en modo *store-and-forward*, NC y el correspondiente al TCP tradicional sobre rutas de 1 y 2 saltos.

Se puede apreciar que ambas técnicas RLC se comportan de forma muy similar cuando $FER = 0$, punto que se corresponde realmente con el experimento anterior. A medida que la FER aumenta, las curvas se separan progresivamente, lo que quiere decir que RLNC completa la información que no llega al destino a través de la ruta de dos saltos, con los paquetes recibidos directamente desde el origen. Esto es posible ya que contienen información diferente. En cambio, con *store-and-forward* la información que se recibe, en muchos casos, estará duplicada en el destino. De ahí la diferencia de rendimiento entre ambas técnicas, que llega a alcanzar un 20 % con $FER = 0.3$.

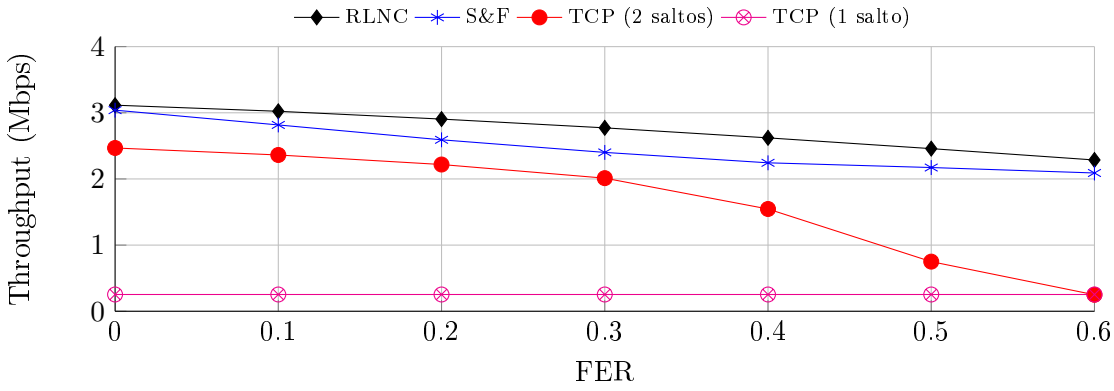


Figura 5.17: Throughput en un canal con pérdidas

Por otro lado se observa que las curvas de TCP quedan muy por debajo de la codificación aleatoria. En el caso de 1 salto, se aprecia un *throughput* constante (0.25 $Mbps$) para todos los valores de FER, ya que todos paquetes se envían a través del enlace directo $S \rightarrow R$, cuya FER es siempre 0.6. Precisamente este valor coincide con el throughput que se puede ver en la Figura 5.6 en el punto $FER = 0.6$.

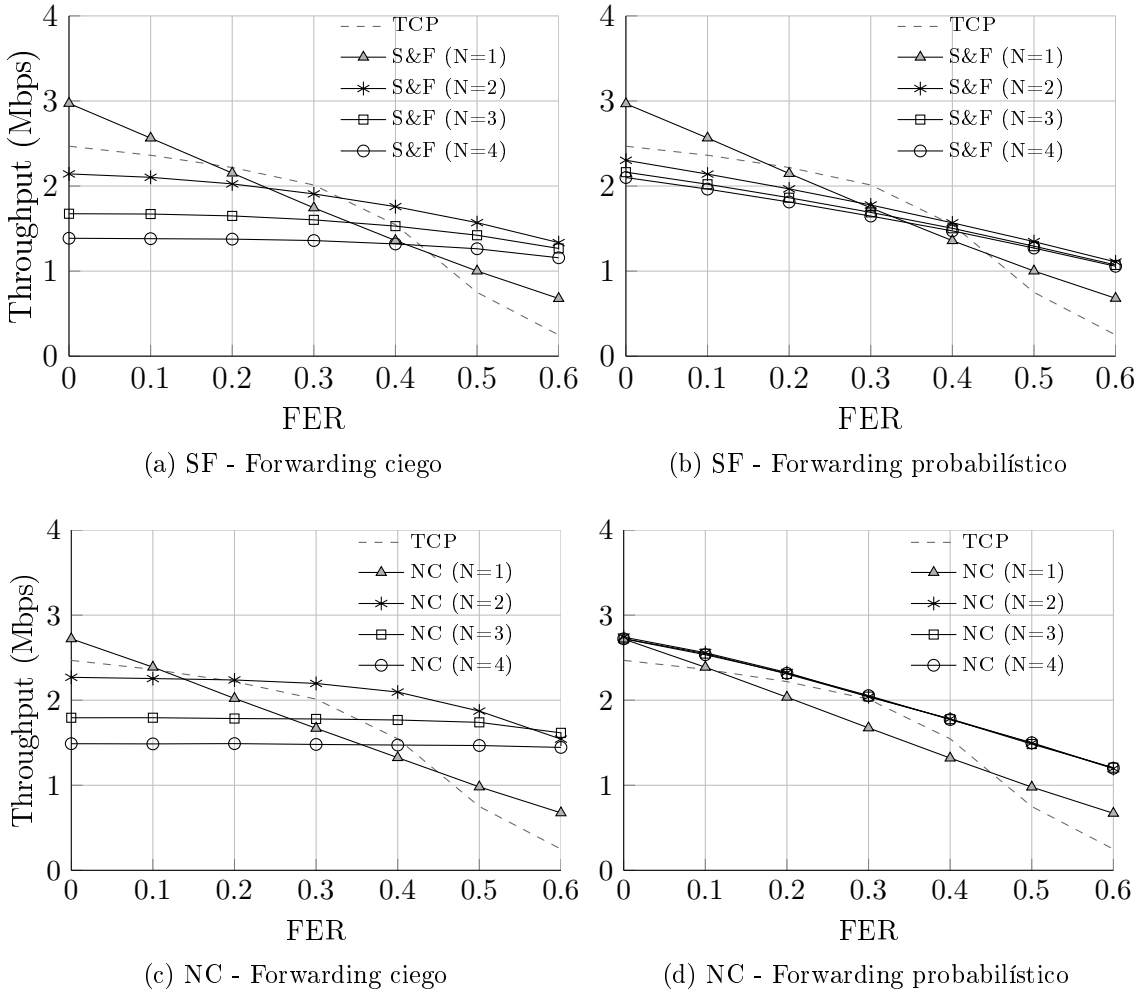
En el escenario de 2 saltos, TCP mantiene un valor ligeramente inferior al RLC hasta que la calidad del enlace $R \rightarrow D$ es menor ($FER > 0.3$). A partir de ese punto, su rendimiento decae rápidamente, proporcionando RLC una ganancia de hasta el 55 %, con $FER = 0.4$, y del 200 % con $FER = 0.5$.

Con esto queda constatado el pobre rendimiento que ofrece TCP en medios inalámbricos a medida que empeora su calidad. También permite identificar las cualidades de las técnicas RLC y NC en dichos canales.

5.4.3. Escenario multi-nodo

En última instancia se pretende analizar los efectos de las técnicas estudiadas hasta este punto sobre topologías multi-camino (una de las características principales de las WMN). Se plantea un escenario donde existen múltiples nodos intermedios, que retransmiten los paquetes recibidos del origen al destino. Cada uno de estos enlaces representaría una ruta diferente en un escenario multi-camino.

Se van a analizar distintas configuraciones para determinar cual de ellas es la mejor. En primer lugar, se hace un estudio del throughput para escenarios con distinto número de nodos intermedios (Figura 5.3). En ellos, se variará el valor de FER de los enlaces desde los nodos intermedios al destino ($R_i \rightarrow D$). La FER que existe en los canales desde el origen a los R_i se fijan a 0, deshabilitando además las funciones de *overhearing*. Se estudiará el rendimiento con la modalidad de *store and forward* tradicional y con NC.



(a) SF - Forwarding ciego

(b) SF - Forwarding probabilístico

(c) NC - Forwarding ciego

(d) NC - Forwarding probabilístico

Figura 5.18: Comparación de cuatro escenarios de retransmisión multinodo

Por otro lado, se distingue entre utilizar un método de retransmisión "ciego", en el que los nodos intermedios retransmiten cada vez que reciben un paquete; y uno probabilístico, en el cual las retransmisiones se realizan con una probabilidad $\frac{1}{N}$, siendo N el número de nodos intermedios. En la Figura 5.18 se pueden apreciar los resultados obtenidos.

Lo primero que se puede observar es el pobre rendimiento que proporciona *store and forward* en este escenario (Figura 5.18a), incluso por debajo de TCP. La razón es que las retransmisiones que realiza cada nodo presentan exactamente la misma información, con lo que la disputa por el canal genera un retraso de las transmisiones de paquetes que van a ser descartados directamente en el destino. Lógicamente, conforme aumenta N, este efecto es más pronunciado. El ejemplo más claro de esto se destaca en la Figura 5.18b. El escenario es el mismo, pero esta vez los nodos retransmiten con una probabilidad menor que 1, con lo que la contienda por el medio físico no agrava tanto el rendimiento de la transmisión.

Sin embargo, a medida que aumenta la FER se puede apreciar que las curvas de $N = 1$ decaen mucho más rápido que las demás. Con $FER = 0$ la mejora es de un 17 %, mientras que, a partir de $FER = 0.2$, el *throughput* es un 15 % menor. La causa es que la redundancia de los nodos intermedios no es suficiente para garantizar la entrega de un paquete concreto. Para $N's$ superiores la probabilidad de que el paquete se entregue aumenta considerablemente, a pesar del valor de la FER. Con la configuración *store-and-forward* "ciego", esto provoca que las curvas, a medida que N crece, tiendan a ser constantes, ya que cada paquete se entrega a través de un nodo u otro.

Por otro lado, en el caso probabilístico, el rendimiento no se ve penalizado por la disputa por el canal, pero decrece más rápidamente debido a la falta de redundancia.

Una vez se introducen técnicas de NC en este escenario, se pueden apreciar las mejoras que proporciona frente al paradigma de *store-and-forward*. Las curvas mantienen formas similares a las vistas en S&F. Sin embargo, se produce una mejora, propiciada por el hecho de que los paquetes que reenvían los nodos intermedios no contienen la misma información, con lo que esa redundancia que antes podía perjudicar al rendimiento, ahora puede aportar información adicional.

En el caso probabilístico se logra que, para valores de N mayores que 2, el rendimiento de NC sea siempre superior al mostrado por TCP (Figura 5.18d). Al comparar ambos métodos (S&F y NC), se puede apreciar una ganancia en el *throughput* de hasta el 25 % para FER bajas y de 17 % para los valores de FER más altos. Por otro lado, en el caso de NC "ciego" (Figura 5.18c) se aprecia también una mejora frente a S&F.

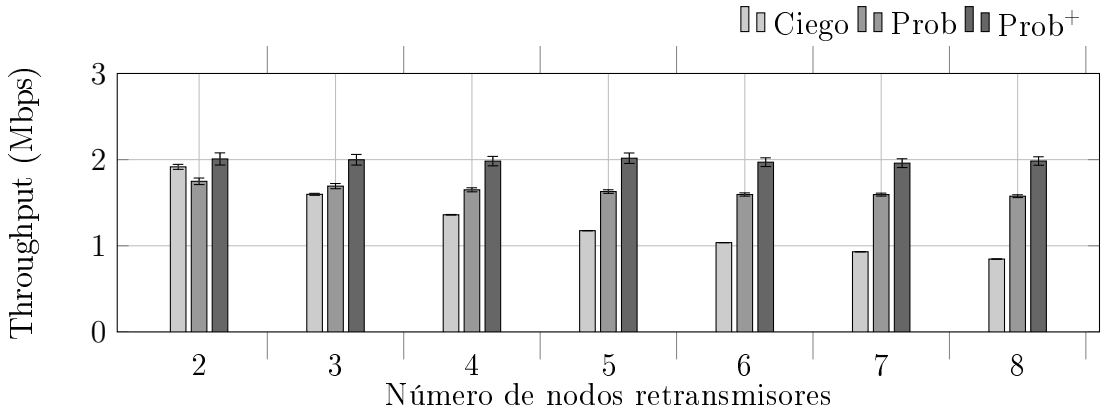
En conclusión, la Figura 5.18 constata las mejoras que introduce tanto NC como la modalidad de transmisión probabilística en entornos multi-camino. Se mitiga el efecto de la contienda por el canal de los nodos intermedios y la redundancia de información transmitida por ellos.

En el siguiente bloque de simulaciones se plantea la introducción de dos técnicas para mejorar el rendimiento. La primera de ellas es la recuperación del *overhearing* visto en el escenario de tres nodos. La segunda es un método de transmisión en los nodos intermedios probabilístico *cross-layer*. En él, la probabilidad de que un nodo transmita el paquete está ponderada por la calidad del enlace del propio nodo con el destino. Se realiza de tal forma que la probabilidad total (entre todos los nodos) de que se transmita un paquete sea igual a 1. Para poder reproducir un escenario en que esto sea posible se establece que la FER de los enlaces $R_i \rightarrow D$ sea aleatoria, situándose entre 0 y 0.6. La FER del enlace directo se establece en 0.6.

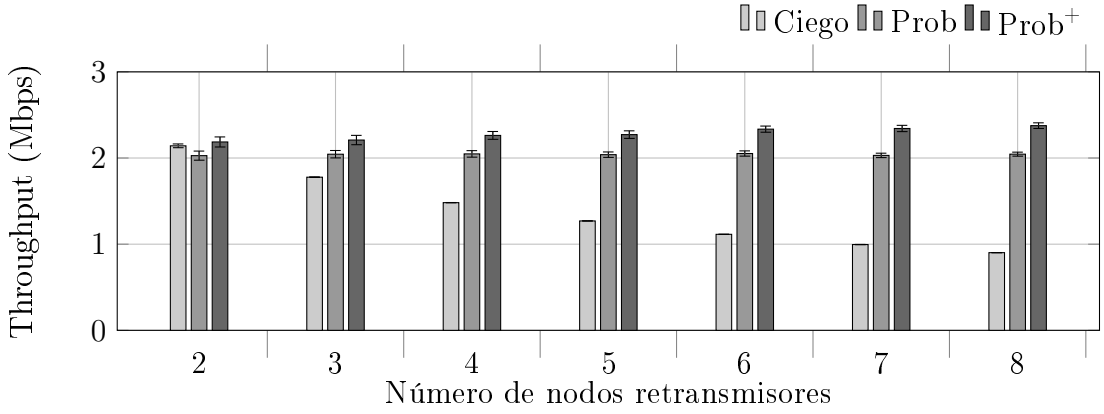
En esta ocasión, se estudian escenarios de hasta 8 nodos intermedios, con las distintas modalidades de transmisión, y variando entre *store and forward* y NC. Los resultados se pueden observar en las Figuras 5.19 y 5.20.

Se puede apreciar la misma tendencia que se ha visto en la Figura 5.18. En la moda-

lidad "ciega" el rendimiento decae rápidamente a medida que aumenta el número de nodos retransmisores, debido a la disputa por el canal.



(a) Store and Forward



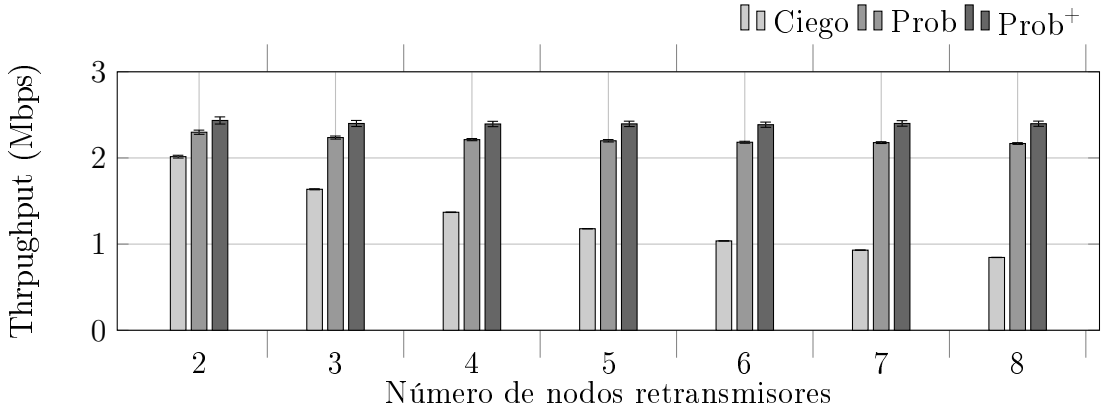
(b) Network Coding

Figura 5.19: Escenarios con *overhearing* deshabilitado

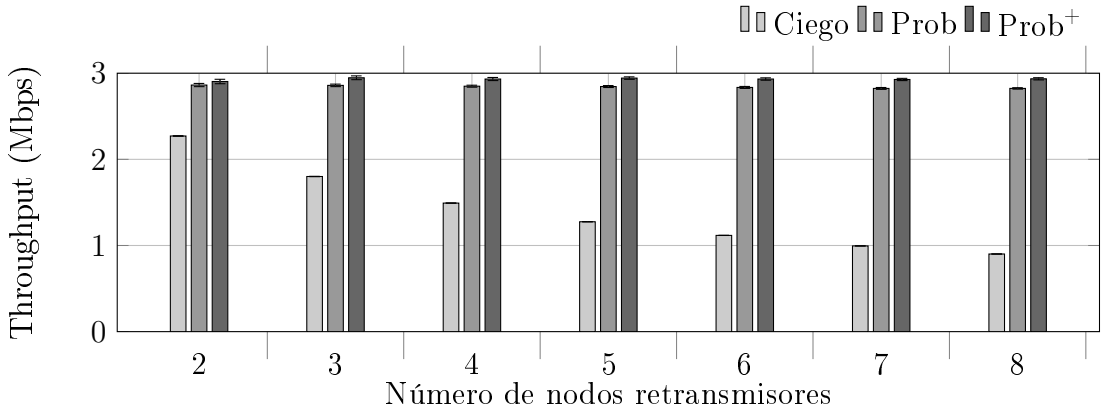
Por otra parte, el *throughput* se mantiene en valores casi constantes para ambas modalidades de envío probabilístico, conforme crece el número de nodos intermedios. Esto implica que, a partir de cierta cantidad de caminos no se produce una ganancia significativa, ya que la llegada de los paquetes está asegurada por alguno de ellos.

El *overhearing* de paquetes, al igual que en el escenario de tres nodos, produce una mejora significativa en todas las modalidades, desde el *forwarding* "ciego" al modelo probabilístico *cross-layer*. En el modelo de S&F se producen ganancias del 37.5 % cuando se transmite con el modelo probabilístico simple y del 20 % para el modelo *cross-layer*. Por otro lado, si se está trabajando con NC la ganancia puede ser del 40 % con la transmisión probabilística y, de nuevo, un 20 % con *cross-layer*. Por tanto, se puede deducir que esta técnica produce mejoras notables en conjunción con una transmisión ordenada en las múltiples rutas, especialmente cuando no se dota de mayor relevancia a ninguno de los

nodos intermedios. El modelo probabilístico, debido a que no distingue si las condiciones son suficientemente buenas para transmitir, tiene mayor margen de mejora.



(a) Store and Forward



(b) Network Coding

Figura 5.20: Escenarios con *overhearing* habilitado

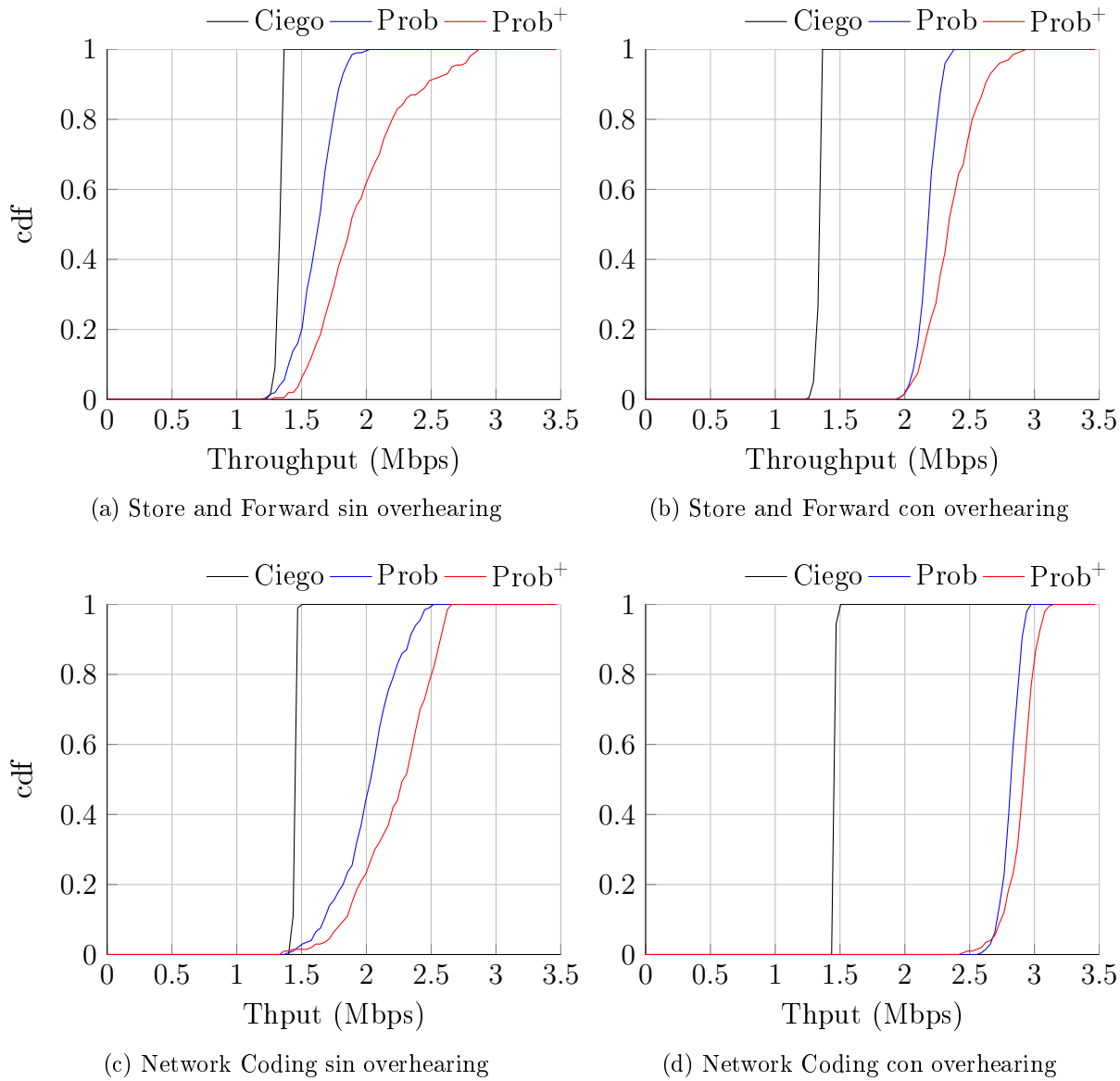
En última instancia, se va a analizar el efecto de monitorizar los enlaces $R_i \rightarrow D$ antes de transmitir un paquete. En este caso, se espera que se aprovechen al máximo las capacidades del canal, y se produzca una mejora en el rendimiento respecto al modelo de transmisión probabilístico simple.

La mejora se aprecia en mayor medida en los escenarios sin *overhearing*. Se alcanza una ganancia del 25 % si se prescinde de la codificación de red y de un 20 % en el caso contrario. En cambio, en escenarios con *overhearing*, la mejora es menor, siendo 9.5 % para S&F y 3.5 % para NC.

Se concluye que existe un límite máximo para este tipo de escenario, como se puede ver en la Figura 5.17. Los casos con rendimientos más altos tienen una posibilidad de mejora menor, mientras que en la configuración S&F sin *overhearing* el margen de mejora es lo suficientemente significativo como para conseguir una ganancia importante.

Este tipo de modalidad de transmisión permite que los nodos intermedios hagan un uso más eficiente del canal, impidiendo que se ocupe el medio con transmisiones que es poco probable sean exitosas a causa de la FER del enlace. Además, se trata de evitar que haya transmisiones simultáneas para que la disputa por el canal sea mínima.

En consecuencia, se puede afirmar que las dos técnicas introducidas en este escenario provocan mejoras considerables en el rendimiento de la transmisión.



(a) Store and Forward sin overhearing

(b) Store and Forward con overhearing

(c) Network Coding sin overhearing

(d) Network Coding con overhearing

Figura 5.21: cdf's de los escenarios con 4 nodos intermedios

Por último, se va a estudiar como se distribuye la cdf del *throughput* de uno de los escenarios anteriores, en concreto el correspondiente a 4 nodos intermedios. Para ello, como

se comentó anteriormente, se realiza un experimento de Monte Carlo con 200 simulaciones independientes. Gracias a esto se podrá realizar el análisis estadístico deseado. Los resultados obtenidos se muestran en la Figura 5.21.

En primer lugar se puede apreciar como la modalidad de retransmisión "ciega" es incapaz de proporcionar un *throughput* superior a 1.5 *Mbps* en cualquiera de los escenarios. Además, la pendiente de subida de la curva es muy cercana a un escalón, lo que indica que la varianza de los resultados es muy pequeña.

Al igual que se ha podido observar en los escenarios anteriores, las curvas relativas a NC alcanzan valores más cercanos a 1.5 *Mbps*, en el caso "ciego", con una ligera ventaja de la configuración correspondiente al escenario con *overhearing* habilitado.

Por otro lado, el método probabilístico simple supone una notable mejora frente a la técnica "ciega". En cualquier caso alcanza al menos los 2 *Mbps*, llegando incluso a asegurar los 2.5 *Mbps* si se combinan RLC, NC y el *overhearing* de paquetes. Este tipo de escenario se beneficia especialmente del *overhearing*, como se puede observar en la Figura 5.21b. En ella se aprecia que, con seguridad, se alcanzan valores de 2 *Mbps* y que el valor máximo obtenido se sitúa alrededor 2.4 *Mbps*. En cambio, el escenario relativo a NC sin *overhearing* (Figura 5.21c) da lugar a unos resultados inferiores. Este despliegue sólo puede asegurar los 2 *Mbps* en un 60 % de los casos, siendo el rendimiento máximo que alcanza de 2.5 *Mbps*. En conjunción con los resultados de análisis previos, el escenario que combina todas las técnicas es el que proporciona el mejor rendimiento, estando siempre dentro del intervalo (2.5 *Mbps*-3 *Mbps*).

La última de las curvas representa el modelo probabilístico *cross-layer*. Esta modalidad termina de mejorar el rendimiento de cada uno de los escenarios. En los casos en los que el *throughput* no tiene valores muy altos, el margen de mejora que aporta es bastante importante, ya que compensa las penalizaciones que provienen de otras fuentes. Por ejemplo, en S&F sin *overhearing* el *throughput* está por debajo de 1.75 *Mbps* en un 80 % de los casos, mientras que con la modalidad probabilística *cross-layer* este valor asciende a 2.2 *Mbps*, quedando 1.75 *Mbps* como valor alcanzado en un 40 % de los experimentos. Además, el máximo rendimiento que se alcanza en una y otra difiere en una cantidad importante, de 1.9 *Mbps* a 2.7 *Mbps*.

Conforme mejora el desempeño de otras técnicas en los escenarios, la ganancia que produce la modalidad *cross-layer* es menor. Esto se puede observar en las Figuras 5.21b y 5.21d, en las que las curvas relativas al envío probabilístico están muy próximas, denotando que la mejora aportada no es tan alta. Se ha de tener en cuenta que el margen para incrementar el rendimiento es menor en estos escenarios.

En conclusión, se ha podido observar como cada una de las técnicas planteadas en este capítulo influyen de una manera positiva en el rendimiento de la transmisión. Se ha estudiado como cada una de ellas condiciona la eficiencia de los escenarios planteados y

que la combinación de todas permite ofrecer un valor muy cercano al máximo alcanzable en los escenarios planteados.

6

Conclusiones y Líneas Futuras

En este último capítulo se recopilan las conclusiones extraídas a lo largo del desarrollo de este trabajo, centrándose, principalmente, en los resultados obtenidos durante las campañas de simulación descritas en el Capítulo 5. En segundo lugar, se exponen una serie de posibles líneas futuras de investigación que quedan abiertas para continuar con la labor realizada en este proyecto.

6.1 Conclusiones

El auge de las comunicaciones inalámbricas exige una continua evolución de las tecnologías relativas a este campo. Uno de los escenarios con más perspectiva de futuro es el de las WMN. Su versatilidad, sencillez de implementación y bajo coste las convierte en una alternativa a tener muy en cuenta. Se hace necesario, por tanto, desarrollar técnicas específicas para aprovechar sus ventajas intrínsecas.En este trabajo se han presentado y analizado RLC y NC.

NC es un campo de estudio bastante reciente y, como tal, existen una serie de incógnitas en torno al rendimiento que puede ofrecer en escenarios reales. Se plantea su uso combinado con UDP, en la capa de transporte, y con RLC. De la investigación realizada sobre estas técnicas se pueden extraer varias conclusiones.

En primer lugar, se ha ha analizado la influencia de los distintos parámetros de RLC en su rendimiento. El número de elementos que componen un fragmento (K) tiene una relación directa con la cantidad de paquetes que se desechan por dependencia lineal en el receptor. Conforme aumenta este valor la probabilidad de recibir paquetes con información útil crece. Sin embargo, a partir de cierto punto ($K = 32$ ó $K = 64$), la mejora proporcionada por este método se estabiliza, pues la cantidad de paquetes que son combinación lineal ya se ha reducido prácticamente al mínimo. El parámetro Q, incrementa la cantidad de posibles valores que puede tomar cada coeficiente. Al igual que K, su aumento reduce

las posibilidades de que se transmitan paquetes linealmente dependientes, sin embargo, su impacto es considerablemente menor. Sólo se aprecia cierta influencia cuando se está trabajando con valores de K bajos. Además, como se ha podido comprobar, la sobrecarga que se introduce en la cabecera cuando Q es alto no es despreciable, pudiendo repercutir en el rendimiento de la transmisión. Por otro lado, tras analizar las librerías de manipulación de cuerpos finitos, se ha determinado que la eficiencia de *FFLAS/FFPACK* es bastante menor que la de *IT++*, con lo que el uso de valores de Q mayores que 1 no compensa, en términos de tiempo de computación. Se concluye por tanto que el uso de un valor K lo suficientemente alto para evitar las combinaciones lineales y uno para Q que no incremente de manera excesiva el tiempo de cómputo (ahorro de batería), es lo más adecuado a la hora de configurar un escenario en el que se utilice RLC.

En la primera campaña de simulaciones se ha podido comprobar que la penalización que provocan los paquetes transmitidos en exceso y los ACK, en un esquema RLC entre dos nodos, no es despreciable y que, en gran medida, se subsana con la elección adecuada de K (suficientemente alto).

Se ha estudiado, asimismo, la evolución del rendimiento de las técnicas RLC según empeora la calidad del enlace, comparándolo con el ofrecido por TCP. Para valores bajos de FER, hasta 0.3, RLC se mantiene ligeramente por encima. Sin embargo, conforme empeora el canal, TCP ve como su rendimiento decae rápidamente (confirmando que no ofrece un buen desempeño en medios de transmisión hostiles). Por otro lado, la caída del *throughput* para RLC es más tendida logrando, así, mejores prestaciones.

También se ha analizado el efecto de las retransmisiones 802.11 sobre el rendimiento de RLC. Teniendo en cuenta que, cada paquete es sustituible por otro cualquiera, en tanto en cuanto la información que tienen es la misma, el mecanismo de recuperación de errores de 802.11 lastra el rendimiento de la transmisión. Debido a esto, se han obtenido *throughput* superiores, al deshabilitar la recuperación de tramas perdidas.

Posteriormente, se introduce un nodo intermedio, con lo que se puede plantear el uso de NC. Esta técnica, combinada con la escucha oportunista de paquetes, produce mejoras frente a un escenario RLSC, ya que las dos posibles rutas por las que se reciben paquetes en el destino proporcionan información distinta. Por contra, S&F genera una redundancia importante de datos en el destino, especialmente cuando la FER del enlace no es muy alta.

Ambas técnicas mejoran el rendimiento del protocolo TCP tradicional en el mismo escenario. Concretamente, se pueden alcanzar ganancias del 200 % con $FER = 0.5$. Por tanto, el hecho de añadir dos posibles fuentes de información (mediante la combinación de NC y *overhearing*) permite superar de manera clara el rendimiento mostrado por TCP.

Por último, se ha planteado el estudio de un escenario en el que varios nodos intermedios reflejan las diferentes rutas de un despliegue multi-camino. Se ha podido observar que las múltiples rutas no producen mejoras al trabajar con con S&F, especialmente cuan-

do el modo de retransmisión es "ciego". Esto se debe a que todos los nodos intermedios retransmiten los mismos paquetes, produciendo una sobrecarga de tramas que no aportan información novedosa. Si se introduce una limitación a la retransmisión de paquetes, el rendimiento aumenta considerablemente, tanto en S&F como en NC. Por ello, se propone el uso de métodos de retransmisión probabilísticos para evitar la sobrecarga del medio inalámbrico.

Con estas técnicas se ha podido apreciar una mejora considerable del rendimiento, sobre todo cuando la calidad del enlace no es muy mala. Su principal característica es que reparte el acceso al canal de una manera ecuánime. Gracias a esto, y en combinación con NC, casi se logra alcanzar el límite de rendimiento que presenta este tipo de escenario, en condiciones ideales.

Dentro de esta técnica, se han propuesto diversas formas para determinar si el nodo ha de retransmitir. En primer lugar, se ha experimentado con un envío equiprobable en todos los nodos. Tras ello, se ha ponderado la probabilidad en función de la calidad del enlace correspondiente. La mejora es evidente, gracias a un mejor reparto del medio físico, especialmente en escenarios donde no es posible realizar la escucha oportunista de paquetes. Sin embargo, en situaciones donde el rendimiento ya ha alcanzado cotas muy altas gracias a NC y al *overhearing*, la mejora no resulta tan apreciable.

En última instancia, se ha analizado la influencia del número de rutas posibles para recibir la información. La conclusión alcanzada es que, una vez que el número de nodos permite que cada paquete llegue al destino, el aumento de estos no deriva en una mejora apreciable en el rendimiento.

Tras todos los estudios realizados, se puede concluir que la combinación de técnicas de NC con RLC y el *overhearing* de paquetes proporciona una mejora notable frente al esquema tradicional basado en TCP. Estas diferencias son mucho más apreciables cuando las condiciones del canal empeoran, constatando que TCP no es un protocolo diseñado para trabajar en entornos hostiles. Por tanto, el impacto de NC en este tipo de redes es importante, pudiendo convertirse en una alternativa muy a tener en cuenta para futuros trabajos, investigaciones o despliegues.

Finalizando, cabe resaltar que los resultados obtenidos a lo largo de este trabajo se han aprovechado para realizar tres artículos de investigación enviados a conferencias de ámbito internacional. Los dos primeros ya han sido aceptados, y el último está en proceso de revisión.

- **Reliable Communications over Lossy Wireless Channels by means of the Combination of UDP and Random Linear Coding** para *"The 19th IEEE Symposium on Computers and Communications"* (ISCC), Madeira, Junio de 2014.

- **Reliable Communications over Wireless Mesh Networks with Inter and Intra-Flow Network Coding** para el *"Workshop on ns-3"* (WNS3), Atlanta, Mayo de 2014.
- **Opportunistic Random Linear Source/Network Coding: Reliable Communications over Wireless Mesh Networks** para el *"IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications"* (PIMRC), Washington, Septiembre de 2014.

6.2 Líneas futuras

En esta sección se introducen algunas posibles líneas futuras de investigación que quedan abiertas para continuar con el trabajo realizado a lo largo de este proyecto.

Cabe destacar la naturaleza *open source* del código desarrollado. Al igual que cualquier aportación realizada al repositorio de ns-3, su función es complementar el código original incluido con el simulador. Por ello, está disponible para que cualquier investigador que lo desee pueda realizar las pruebas que estime oportunas, ampliando el estudio desarrollado en este trabajo.

Como se ha comentado anteriormente, las técnicas de NC son bastante recientes y se encuentran en un estado inicial en su desarrollo. Aún existe cierta controversia sobre la viabilidad de estas soluciones en un entorno de red real. Por tanto, queda mucho camino por recorrer, tanto en la investigación de nuevas técnicas, como en evolución de las ya conocidas.

El primer punto se origina tras detectar un pequeño fallo de la librería *FFLAS/FFPACK*, que no permite trabajar correctamente con cuerpos finitos de orden mayor que 2. Sería fundamental solventarlo si se pretende trabajar con cuerpos distintos a los utilizados en este trabajo.

Otro aspecto interesante sería buscar nuevos esquemas de codificación, diferentes a RLC, que pudieran aportar una mejora del rendimiento. En estos casos, tendrían que implementar de alguna forma las funcionalidades que se pierden al eliminar TCP como protocolo de transporte. Además, resulta interesante plantear futuras campañas de simulación con escenarios generados aleatoriamente, para comprobar la viabilidad de NC en un escenario real. En estas topologías podría resultar de utilidad seleccionar los nodos que ejercerán las tareas de recodificación de paquetes dentro de la red. El diseño de un procedimiento para seleccionar estos nodos dinámicamente, en función de las características de la red, es otro de los aspectos que sería interesante afrontar en el futuro.

En escenarios en los que existen flujos de datos de diversas fuentes, puede resultar de

gran interés integrar las dos modalidades de NC vistas en este trabajo. El desarrollo de una plataforma que las combine para lograr mejoras adicionales a las vistas en el Capítulo 5 es una línea a tener en cuenta.

Siguiendo con algunos de los trabajos realizados por el Grupo de Ingeniería Telemática (GIT), se podría plantear aprovechar las características multi-camino de las redes malladas para que la transmisión se realice por diferentes interfaces. Esto permite que un nodo transmita o reciba por varios "caminos" independientes al mismo tiempo, aumentando el *throughput* total.

Por último, estas técnicas podrían ser implementadas sobre redes de sensores de dispositivos. Por su topología intrínseca de red mallada, aparece como una de las tecnologías en las que estas técnicas podrían aportar beneficios. Para ello, sería importante estudiar la viabilidad de implementar estos esquemas en dispositivos reales, especialmente considerando las limitaciones que presentan computacionalmente.

Bibliografía

[1] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204 –1216, July 2000.

[2] R. Stewart. Stream Control Transmission Protocol. RFC 4960(Proposed Standard), September 2007.

[3] S. Katti, H. Rahul, Wenjun Hu, D. Katabi, M. Medard, and J. Crowcroft. XORs in the Air: Practical Wireless Network Coding. *Networking, IEEE/ACM Transactions on*, 16(3):497 –510, 2008.

[4] M. Hundebøll, J. Ledet-Pedersen, J. Heide, M.V. Pedersen, S.A. Rein, and F.H.P. Fitzek. Catwoman: Implementation and performance evaluation of ieee 802.11 based multi-hop networks using network coding. In *Vehicular Technology Conference (VTC Fall), 2012 IEEE*, pages 1–5, 2012.

[5] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. Trading structure for randomness in wireless opportunistic routing. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '07, pages 169–180, New York, NY, USA, 2007. ACM.

[6] J.K. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros. Network Coding Meets TCP: Theory and Implementation. *Proceedings of the IEEE*, 99(3):490 –512, march 2011.

[7] H. Seferoglu, A. Markopoulou, and K.K. Ramakrishnan. I2nc: Intra- and inter-session network coding for unicast flows in wireless networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1035–1043, 2011.

[8] J. Krigslund, J. Hansen, M. Hundeboll, F.H.P. Fitzek, and T. Larsen. CORE: COPE with MORE in Wireless Meshed Networks. In *IEEE VTC2013-Spring: Cooperative Communication, Distributed MIMO and Relaying*, Dresden, Germany, June 2013.

[9] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. MORE: A Network Coding Approach to Opportunistic Routing. Technical report, Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, June 2006.

[10] Sanjit Biswas and Robert Morris. Opportunistic routing in multi-hop wireless networks. *SIGCOMM Comput. Commun. Rev.*, 34(1):69–74, January 2004.

[11] ns3-YANCI. Yet Another Network Coding Implementation, 2012. https://github.com/yangchi/ns3-yanci.

[12] Diogo Ferreira, Luísa Lima, and João Barros. Neco: Network coding simulator. In Olivier Dalle, Gabriel A. Wainer, L. Felipe Perrone, and Giovanni Stea, editors, *SimuTools*, page 52. ICST, 2009.

[13] Morten V. Pedersen, Janus Heide, and Frank H. P. Fitzek. Kodo: An open and research oriented network coding library. In *Proceedings of the IFIP TC 6th International Conference on Networking*, NETWORKING'11, pages 145–152, Berlin, Heidelberg, 2011. Springer-Verlag.

[14] O. Trullols-Cruces, J.M. Barcelo-Ordinas, and M. Fiore. Exact decoding probability under random linear network coding. *Communications Letters, IEEE*, 15(1):67–69, 2011.

[15] IEEE Standard for Information technology–Telecommunications and information exchange between systems Local and metropolitan area networks–Specific requirements Part 11: Wireless LAN medium access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1 –2793, 29 2012.

[16] R. Oarroppo, S. Giordano, and S. Lucetti. Ieee 802.11 b performance evaluation: convergence of theoretical, simulation and experimental results. *Telecommunications Network Strategy and Planning Symposium. NETWORKS 2004, 11th International*, 2004.

[17] The ns-3 network simulator. http://www.nsnam.org/.

[18] David Gómez, Eduardo Rodríguez, Mario Puente, and Ramón Agüero. Network coding architecture source code and documentation (ns-3). https://github.com/dgomezunican/network-coding-ns3.

[19] M. Lacage and T.R. Henderson. Yet another network simulator. *WNS2 '06 Proceeding from the 2006 workshop on ns-2: the IP network simulator*, 2006.

[20] IT++ Mathematical library. http://itpp.sourceforge.net/.

[21] FFLAS-FFPACK. Finite Field Linear Algebra subroutines package. http://www-ljk.imag.fr/membres/Jean-Guillaume.Dumas/FFLAS/index.html.

Acrónimos

ACK *Acknowledgment*

B_S *Buffer Size*

C_T *Coding Time*

CATWOMAN *Coding Applied To Wireless On Mobile Ad-Hoc Networks*

cbr *constant bit rate*

cdf *cumulative distribution function*

CW *Contention Window*

COPE *Opportunistic Coding in Practical Wireless Network Environment*

FER *Frame Error Rate*

IP *Internet Protocol*

LBNL *Lawrence Berkeley National Laboratory*

MAC *Media Access Control*

MIT *Massachusetts Institute of Technology*

MTU *Maximum Transfer Unit*

NC *Network Coding*

ns-3 *Network Simulator 3*

OTcl *Object Tcl*

QoS *Quality of Service*

RLC *Random Linear Coding*

RLNC *Random Linear Network Coding*

RLSC *Random Linear Source Coding*

SCTP *Stream Control Transmission Protocol*

Tcl *Tool Command Language*

TCP *Transmission Control Protocol*

UDP *User Datagram Protocol*

WMN *Wireless Mesh Networks*

XOR *Exclusive OR*

YANCI *Yet Another Network Coding Implementation*