ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

*Proyecto Fin de Carrera*

# Mejora del comportamiento del protocolo UDP sobre redes inalámbricas multi-salto a través de técnicas de Network Coding

## (Improvement of UDP protocol behavior over wireless networks through Network Coding techniques)

Para acceder al Titulo de

**INGENIERO DE TELECOMUNICACIÓN**

Autor: Mario Puente Pomposo

Octubre - 2013

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

# INGENIERÍA DE TELECOMUNICACIÓN

CALIFICACIÓN DEL PROYECTO FIN DE CARRERA

**Realizado por:** Mario Puente Pomposo
**Director del PFC:** Ramón Agüero Calvo, David Gómez Fernández
**Título:** "Mejora del comportamiento del protocolo UDP sobre redes inalámbricas multi-salto a través de técnicas de Network coding"
**Title:** "Improvement of UDP protocol behavior over wireless networks through Network Coding techniques"

**Presentado a examen el día:** 31 de Octubre de 2013

para acceder al Título de

**INGENIERO DE TELECOMUNICACIÓN**

Composición del Tribunal:
Presidente (Apellidos, Nombre): García Arranz, Marta
Secretario (Apellidos, Nombre): Agüero Calvo, Ramón
Vocal (Apellidos, Nombre): Vía Rodríguez, Javier

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

# Índice de Figuras

# Resumen ejecutivo

En la actualidad existen multitud de técnicas y mecanismos para mejorar el rendimiento en redes de comunicación cableadas. Sin embargo, en la última década, la repercusión de los sistemas inalámbricos obliga a satisfacer la demanda de una calidad cada vez mayor por parte de los usuarios, intentando alcanzar un comportamiento comparable al de las redes cableadas.

En este documento se presenta un análisis de la mejora sobre comunicaciones basadas en tráfico UDP al trabajar con técnicas de codificación de red (Network Coding) sobre redes inalámbricas multi-salto. Los nodos intermedios realizan la codificación de información de varios flujos, para enviarla hacia sus destinos correspondientes, logrando un incremento del rendimiento de la transmisión. Los avances tecnológicos en materia de codificación pretenden, además, obtener mejoras respecto a la seguridad de la comunicación.

Para analizar los beneficios y problemas de la aplicación de estas técnicas, en este proyecto se ha hecho uso de la herramienta NS-3. Tras la implementación de la nueva entidad, *Network Coding*, y realizando un alto número de simulaciones sobre varios escenarios, se ha observado un incremento del rendimiento en torno al 50 % en canales ideales.

# Abstract

At present there are several technics and mechanisms to improve efficiency on wired communication networks. However, during last decade, the repercussion of wireless systems obliges to satisfy an increasing demand of quality from users, in an attempt of reaching behaviour comparable to wired networks.

This document portrays an analysis of the enhancement of communications based on UDP traffic when working using Network Coding techniques over wireless mesh networks. Intermediate nodes make the coding of various flows information, to send it to corresponding destination, achieving a transmission capacity upturn. In addition, technological progress in matters of coding expects to obtain improvements in communication security .

For the purpose of evaluating the advantages and disadvantages of applying these technics, the NS-3 tool has been used. Following the implementation of the new identity, Network Coding, and making numerous simulations over different scenarios, it has been observed an increase of performance around 50 % in ideal channels.

# Agradecimientos

Quiero expresar mi agradecimiento a Ramón Agüero, bajo cuya dirección se ha realizado este trabajo, por su incondicional apoyo y asesoramiento en todos los aspectos de esta investigación y elaboración de este proyecto, así como por la confianza depositada en mí.

A David Gómez por su inestimable colaboración en el desarrollo, implementación y análisis de los resultados, así como por su constancia en este trabajo y su apoyo a nivel personal.

A todos mis amigos, que tantos momentos buenos me han brindado durante estos años de carrera y amenizado los no tan buenos.

A toda mi familia, especialmente a mis padres y hermanos, que constituye un pilar fundamental en mi vida.

A mis abuelos, que aunque no están presentes, les dedico este trabajo.

A todas las personas que de una forma u otra han contribuido a que este proyecto tenga su fin.

# 1 Introducción

## 1.1 Planteamiento del problema

El presente proyecto de investigación consiste en analizar las técnicas de Network Coding sobre redes malladas inalámbricas (WMN), empleando un protocolo de transporte no orientado a conexión. Hoy en día, el uso de estas redes se ha extendido a todos los ámbitos de nuestra vida vida cotidiana ya que, gracias a las grandes posibilidades de movilidad, las convierte en uno de los campos de mayor crecimiento en los últimos años. Pero no todo son ventajas, ya que el medio de propagación que utilizan estas redes es el aire, por lo que las señales pueden ser interceptadas, lo que se traduce en una comunicación no segura, a pesar de que las técnicas de autenticación y de codificación que se han introducido en los últimos tiempos han disminuido este inconveniente. Debido a las largas distancias entre los nodos de la red o a la aparición de interferencias, traducidas como errores en el canal pueden aparecer problemas en la recepción de paquetes.

A lo largo de este proyecto se ha trabajado con un protocolo de transporte denominado UDP, con el que se esperan visualizar mejoras en el rendimiento de la red, debido a la ausencia total de un control de congestión como protocolo no orientado a conexión que se trata.

Las técnicas de *Network Coding* que han surgido en los últimos años pretenden minimizar parte de los problemas mencionados. Se aprovechan de muchas de las características de las redes inalámbricas para introducir mejoras, tanto en el rendimiento de la red como en la seguridad de las comunicaciones.

## 1.2 Objetivos

El principal objetivo de este proyecto es el de llevar a cabo un estudio detallado acerca de las mejoras que introducen las técnicas de *Network Coding* sobre tráfico UDP en redes inalámbricas multi-salto. Para este cometido se ha utilizado como herramienta para desarrollar las simulaciones *Network Simulator 3*. Para analizar la influencia de NC se hará uso de tres topologías muy sencillas, sobre las que se cuantifican las ventajas de NC sobre el rendimiento de la red.

Se analizará la influencia de diferentes parámetros de operación de NC, como el

número de paquetes que puede almacenar un nodo, la situación de un nodo respecto a otro, el número de saltos, el retardo, la calidad del canal, etc sobre el rendimiento total de la red.

Finalmente, como se ha adelantado se ha llevado a cabo el desarrollo de la implementación de NC en el marco del simulador NS3 ya que, por defecto, no incluye los módulos necesarios para reproducir los procesos de codificación y decodificación de paquetes que se emplean. Debido a la multitud de simulaciones que se llevan a cabo se automatiza en la mayor medida posible los procesos de medida.

## 1.3 Estructura de la memoria

El documento se encuentra estructurado en los capítulos que se introducen seguidamente.

En el Capítulo 2 se recogen los aspectos teóricos principales que servirán de apoyo para seguir adecuadamente los temas tratados a lo largo de la memoria. Se abarcarán las Wireless Mesh Networks, la técnica de Network Coding y, finalmente, se hará referencia a algunos estudios precendetes.

En el Capítulo 3 se describe la plataforma que se ha empleado para la realización de las simulaciones llevadas a cabo, Network Simulator 3. Se presentan sus características, componentes, así como la implementación realizada de los módulos de Network Coding.

En el Capítulo 4 se profundiza en el funcionamiento de Network Coding en el simulador NS3 desde un punto de vista funcional: sus procesos principales (codificación y decodificación), la cabecera de esta nueva entidad, el funcionamiento de los buffers internos, etc. Se detallará el formato de los archivos empleados para la realización de una simulación automatizada.

En el Capítulo 5 se presentan los parámetros que se han recogido a lo largo de todas las simulaciones realizadas, junto a una breve descripción de la toma de decisiones común a todas ellas. Se explicarán las topologías que se han empleado, así como la operación de NC sobre ellos. Al final, se destacarán los resultados más interesantes, para comprender los beneficios de NC.

En el Capítulo 6 se sintetizan los resultados del trabajo tras los análisis que se han llevado a cabo en el capítulo anterior. Se proponen además las posibles líneas de investigación que el empleo de NC sobre UDP deja abiertas para el futuro.

# 2
# Estado del arte

Este capítulo presenta la introducción a cada uno de los aspectos principales que componen el trabajo, para que el lector pueda adquirir unos conocimientos básicos que le permitan comprender en mayor medida los diferentes asuntos tratados a lo largo de las siguientes páginas.

## 2.1 Wireless Mesh Networks

Las redes malladas inalámbricas (WMN) aparecen como un posible camino en la evolución de las comunicaciones inalámbricas, ya que permiten conectarse a la red a dispositivos a pesar de estar fuera del rango de cobertura de los elementos de acceso. Se puede pensar, por ejemplo, en interconectar varios puntos de acceso WI-FI y formar una malla, que proporcione una amplia cobertura. Los nodos son capaces de establecer una comunicación entre ellos si sus zonas de cobertura se solapan entre sí. Con esta idea básica se pueden desplegar redes robustas, rápidas y flexibles, además de configurables, sin necesidad de la costosa infraestrcutura cableada.

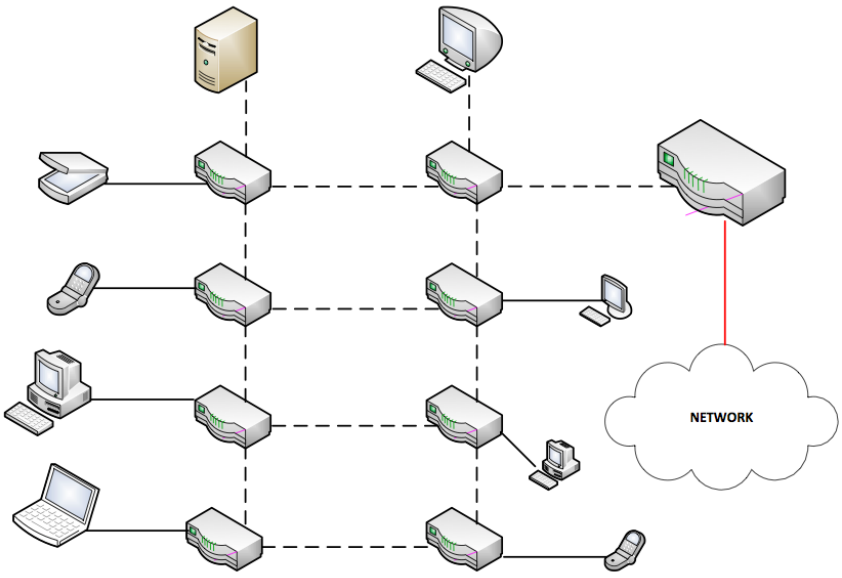



Figura 2.1: Red mallada inalámbrica

WMN ofrece las siguientes ventajas:

- Bajo coste de instalación y mantenimiento respecto a la red cableada.
- Despliegue rápido (cuestión de horas).
- Facilidad para cubrir aquellas zonas que son de difícil acceso para cablear.
- Balanceo de la carga de tráfico y mejora de la tolerancia a fallos, ya que si un nodo cae, la red puede autorreconfigurarse para encontrar otras rutas alternativas de acceso.

Sobre esta tecnología se sustentan los escenarios que se han analizado para caracterizar el comportamiento de las técnicas de Network Coding sobre UDP.

## 2.2 Network Coding

La técnica conocida como *Network Coding*, cuya base es el núcleo de este trabajo, ha sido el foco de numerosas investigaciones, esfuerzos y propuestas, ya que ofrece mejoras apreciables, tanto en rendimiento como en seguridad. Fue Ahlswede quien demostró el aumento de la capacidad cuando los nodos intermedios de la red mezclan la información de distintos flujos en uno solo, a la vez que el receptor cuenta con la informacón necesaria para obtener los datos dirigidos hacia él. En las siguientes páginas se explicarán los principios básicos sobre los que se sustenta esta innovadora técnica, presentando algún ejemplo ilustrativo.

Cuando la información viaja a través de la red en paquetes de datos, las únicas modificaciones que se producen son las necesarias en las diferentes cabeceras, por tanto, los datos no son cambiados entre el emisor y el receptor, siendo flujos totalmente independientes los unos de otros.

En la actualidad, a los nodos intermedios se les puede dar la capacidad de procesar la información de estos paquetes, de forma que pueden unirse diversos flujos de datos, transmitiendolos como uno solo. Por tanto, el nodo intermedio deja de tener un papel de reenvío de información al siguiente nodo, para pasar a ser algo más. Su función principal es precisamente la de combinar la información procedente de diversas fuentes independientes, mientras que el receptor va a ser el único nodo de la red que va a separar tal mezcla para obtener el flujo independiente dirigido hacia él. Se ha analizado que dotar al nodo intermedio de tal capacidad puede ofrecer grandes ventajas, llegando a convertirse en una tecnología que pudiera ser ampliamente utilizada por las redes del futuro.

A continuación, se ilustra con un ejemplo muy trivial el concepto de Network Coding.

El escenario, que posteriormente será analizado en términos de rendimiento, consta de tres nodos, con dos usuarios que pretenden realizar un intercambio de paquetes. La secuencia natural comienza cuando el usuario X envia su primer paquete hacia el nodo

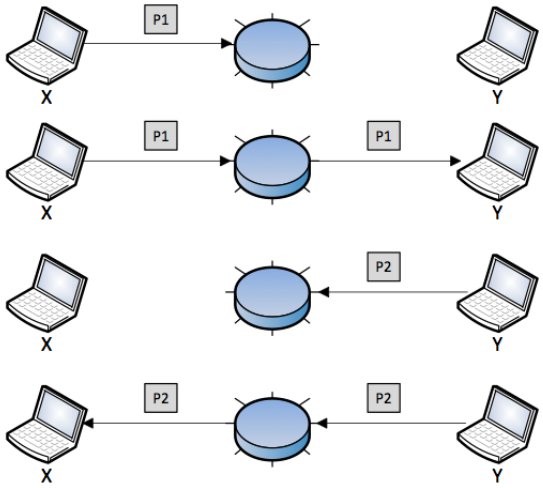



Figura 2.2: Escenario lineal sin *Network Coding*

central. Este lo retransmite hacia el usuario Y, completándose asi el primer envío. Ahora Y lanza su paquete al nodo intermedio, que se lo hace llegar a X. Obsérvese que el número de transmisiones total que se han realizado es cuatro.

A continuación se presentan las características más relevantes de la implementación de Network Coding que ha sido utilizada.

1. *Coding Node*: nodo intermedio de la red, encargado de realizar la codificación de los datos pertenecientes a flujos independientes. El paquete entrante se denominará paquete nativo, mientras que el nuevo será el paquete codificado, cuya transmisión hará uso de la modalidad *pseudo-broadcast*.

   La técnica empleada para la codificación es una sencilla operación XOR bit a bit entre todos los paquetes. Por tanto el receptor, en el momento de filtrar el paquete codificado, tiene que contar con los paquetes nativos, exceptuando el que busca recuperar.

2. *Coding Time*: temporizador que establece el tiempo al que se mantienen los paquetes que se encuentran a la espera de otros nativos, de forma que se incremente la oportunidad de codificaciones y, consecuentemente, el rendimiento. Este reloj es necesario debido a la existencia de retardos aleatorios que se producen, y que podrían minimizar las oportunidades de codificación.

3. *Buffer Size*: parámetro que determina el número de paquetes que pueden almacenarse simultáneamente en el *Coding Node*. A mayor tamaño, mayores oportunidades de codificación y, teóricamente, mayor rendimiento.

4. Proceso Overhearing: característica que permite a los nodos receptores "escuchar" los paquetes que hayan sido transmitidos dentro de su área de cobertura, aunque no vayan dirigidos hacia ellos. Gracias a esta propiedad, el nodo receptor almacena

los paquetes en el *Decoding Buffer*, siendo lo que le permite realizar la posterior decodificación.

A continuación, se aplica Network Coding en el ejemplo anterior. En este caso el nodo central, *Coding Node*, va a convertir distintos flujos independientes en uno solo. Los receptores, gracias a las técnicas de *overhearing* y *pseudo-broadcast* van a poder obtener la información dirigido a ellos.

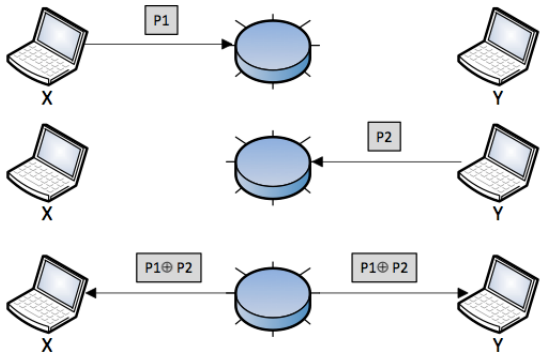



Figura 2.3: Escenario lineal con *Network Coding*

El nodo central se ha transformado en un *Coding Node*, que recibirá los paquetes de los usuarios X e Y, que serán codificados mediante una operación XOR en un único paquete, que será recibido simultáneamente gracias al modo *pseudo-broadcast*. Como los receptores cuentan con el paquete codificado y con el que ellos mismos transmiten, sólo basta con volver a realizar la operación XOR para adquirir el paquete dirigido a ellos. El aspecto a resaltar en este ejemplo tan sencillo es que, respecto al anterior, han realizado tres transmisiones, lo que supone una mejora de la eficiencia energética (menos transmisiones), una reducción del retardo y de las posibles interferencias.

Una vez vistos los beneficios de la aplicación de Network Coding sobre un escenario sencillo, vamos a fijarnos en topologías más complejas y realistas que será a lo que nos tendremos que enfrentar en un futuro.

En cuanto al segundo ejemplo recogido en la Figura 2.4 se trata de una topología conocida como X que, además, será analizada posteriormente en esta memoria. Se dispone de dos transmisores S1 y S2 y dos receptores D1 y D2, donde cada uno de los primeros transmite de manera genérica un bit, denotado como x1 o x2, por cada slot temporal.

La figura de la izquierda muestra el comportamiento del tráfico si los envíos se han realizado solamente a D1. El bit enviado por S1 seguirá el camino S1-D1, mientras que el bit transmitido por S2 seguirá los enlaces S2-R, R-D1. En la segunda figura se muestra una situación similar cuando los envios se dan solamente hacia D2. El tercer ejemplo es el que resulta más interesante, porque pretende que ambos destinos reciban simultáneamentelos datos de S1 y S2. Sin Network Coding, solo uno de los destinos recibirá ambos bits, en función del que se decida transmitir primero por el nodo intermedio, mientras que el

otro se quedaría sin parte de la información. Aquí se pone de manifiesto la capacidad de NC para realizar la codificación de los bits de los nodos intermedios. De forma que el nodo intermedio, *Coding Node*, realiza una operacion XOR sobre x1 y x2 para obtener un nuevo bit x1 $\oplus$ x2. Por tanto, D1 recibirá la dupla (X1, X1 $\oplus$ X2) y D2 obtendrá (X1, X1 $\oplus$ X2). Tras realizar la decodificación, la información llegará simultáneamente a los destinos. Finalmente, es importante destacar que el throughput se ha incrementado por la manipulación intermedia de bits.

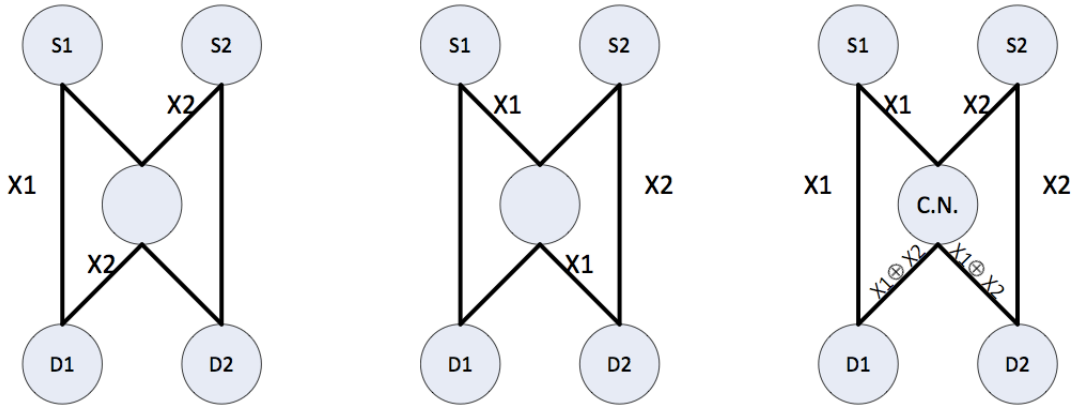



Figura 2.4: Network Coding en topología X

Resaltamos las tres principales ventajas de la aplicación Network Coding sobre las *Wireless Mesh Networks*: maximización del throughput, minimización de la energía por bit y reducción del retardo.

El interés suscitado en toda la comunidad científica es inevitable, atrayendo la atención de matemáticos, ingenieros, etc. Desde su introducción, las técnicas de *Network Coding* han sido pensadas para aumentar el throughput y disminuir el retardo, aunque hoy en dia se ha producido un incremento en el número de aplicaciones que las utilizan. Algunas de ellas se recogen en la Figura 2.5

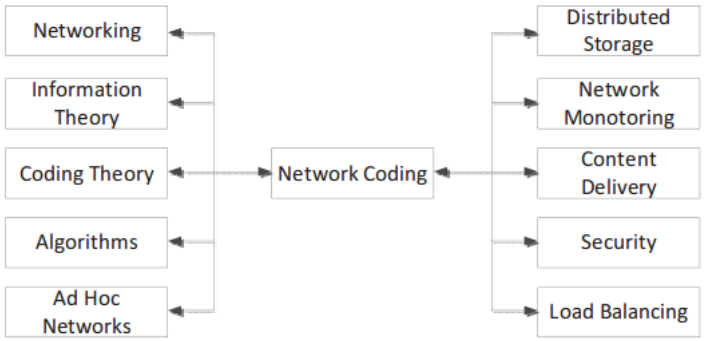



Figura 2.5: Utilización de *Network Coding* en otros ámbitos

### 2.2.1. Problemas

Como se puede observar, los beneficios de la implementación de NC son claros. Por otro lado esta técnica también puede generar conflictos.

A nivel de enlace (MAC 802.11): la transmisión de un paquete codificado se realiza utilizando una modalidad de broadcast modificada, aprovechando varias virtudes de los envíos unicast, ya que un envío broadcast presenta un problema fundamental, y es que un paquete enviado en este modo nunca es retransmitido. Por tanto, si el *Coding Node* enviase los paquetes de esta forma no se estaría ofreciendo ningún tipo de fiabilidad a la transmisión. El paquete tiene que ir dirigido exclusivamente a una única dirección MAC. La solución al problema es sencilla, gracias al proceso de *overhearing* que realizan los nodos receptores, de forma que, aunque el paquete codificado vaya dirigido a un único destino, el resto va a poder recibirlo y procesarlo. Por este motivo, la mayoría de las implementaciones de Network Coding que se analizan hoy en dia utilizan un modelo de envío de paquetes que es una combinación entre unicast y broadcast, denominada *pseudo-broadcast*.

## 2.3 Buffer Size

Dos paquetes son codificados cuando pertenecen a flujos distintos para lo que se mantienen en el *Coding Buffer*; sólo se almacenarán paquetes si está vacío o bien si pertenecen al flujo de los que ya estuvieran almacenados, esperando a una oportunidad de codificación. Existen tres caminos para que un paquete individual o codificado abandone el *Coding Node*, tal y como se describe a continuación.

1. *Buffer Timer*: la cuenta atras del temporizador llega a cero y el paquete, que no ha encontrado su oportunidad de codificación, es transmitido automáticamente.

2. *Buffer Size*: el buffer está lleno y, tras la recepción de un nuevo paquete del mismo flujo, el más antiguo deberá ser transmitido, pasando el más reciente a ocupar su lugar.

3. Oportunidad de Codificación: llega un paquete de flujo diferente al que se encuentra almacenado en el buffer.

En [1] se presenta el modelo con el que se pretende simular la probabilidad de que se obtenga una oportunidad de codificación en función del parámetro *Buffer Size*, mientras que la influencia del temporizador no se tiene en cuenta, considerándolo excesivamente grande. Se tomará como fallo la situación en la que lleguen paquetes pertenecientes al mismo flujo, mientras que el éxito será cuando sean de flujos diferentes, apareciendo así la oportunidad buscada.

Considérese p la probabilidad de que un paquete pertenezca a uno de los flujos, mientras que (1 - p) se corresponde con la que sea del otro, asumiendo, por tanto, dos flujos. Nótese que cuando el buffer se encuentra vacío no puede hablarse ni de un éxito

ni un fracaso para la codificación.

En primer lugar, se tiene un *Buffer Size* para un paquete. Se denominará A al que proceda del primer flujo, mientras que B será un paquete del segundo flujo. Cuando el buffer se encuentra vacío y llega un paquete, no ocurre nada, mientras que si no lo está y se produce la llegada de uno nuevo se pueden dar dos situaciones:

- Éxito en la codificación: si el paquete A se encuentra almacenado y se produce la recepción de otro de distinto flujo, B, o viceversa:

$$P_{exito} = p * (1 - p) + (1 - p) * p \tag{2.1}$$

- Fallo en la codificación: si el paquete A se encuentra almacenado y se produce la recepción de uno del mismo flujo, A, o viceversa:

$$P_{fallo} = p * p + (1 - p) * (1 - p) \tag{2.2}$$

Se incrementará ahora el tamaño del *Buffer Size* para que pueda albergar dos paquetes simultáneamente, de forma que, si llegan dos paquetes procedentes del mismo flujo, serán almacenados automáticamente. Las nuevas probabilidades serán:

$$P_{exito} = p * (1 - p) + (1 - p) * p + p * p * (1 - p) + (1 - p) * (1 - p) * p \tag{2.3}$$

$$P_{fallo} = p * p * p + (1 - p) * (1 - p) * (1 - p) \tag{2.4}$$

De forma que se pueden generalizar las expresiones anteriores para un *Buffer Size* de tamaño N, permitiendo modelar las probabilidades de éxito y fallo en la codificación:

$$P_{exito} = (1 - p) \sum_{i=1}^{N} p^i + p \sum_{i=1}^{N} (1 - p)^i \tag{2.5}$$

$$P_{fallo} = p^{N+1} + (1 - p)^{N+1} \tag{2.6}$$

Se va a emplear un ejemplo en el que exista la misma probabilidad de que un paquete proceda de un flujo u otro, tomando p = 0.5. En la Figura 2.6 se representan las dos probabilidades anteriores en función del *Buffer Size*.

Obsérvese que el aumento del tamaño del *Buffer Size* conlleva también a un incremento de a las oportunidades de codificación. Hay que tener en cuenta, en cualquier caso que se ha considerador el *Buffer Timer* infinito.

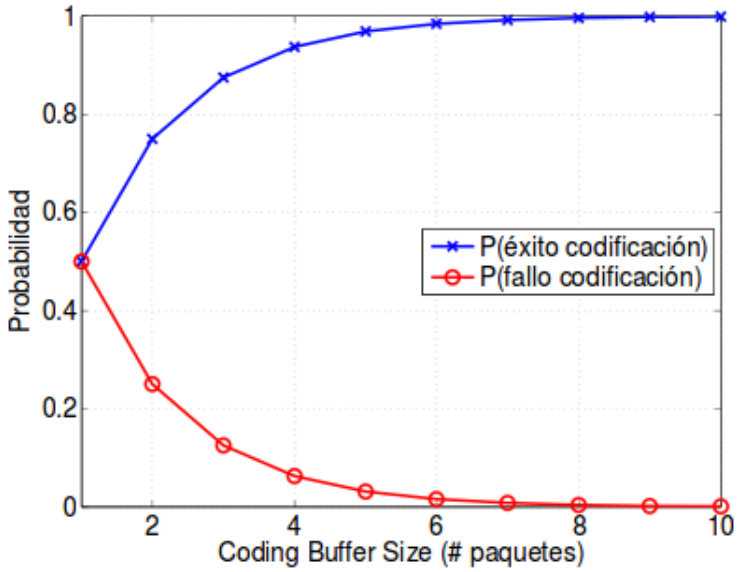



Figura 2.6: Probabilidad de éxito y fallo en la codificación

## 2.4 Nivel de transporte: User Datagram Protocol

Se trata de un protocolo de nivel de transporte con un servicio de transmisión no orientado a conexión, utilizado por aplicaciones en las que no es posible realizar retransmisiones, normalmente por los estrictos requisitos de retardo que se tiene en estos casos.

Entre sus características se pueden resaltar las siguientes:

- La unidad de información se denomina datagrama o mensaje.
- No requiere de un establecimiento explícito de la conexión, por lo que no se introducen retardos.
- No es fiable, ya que no existen reconocimientos (confirmaciones mediante ACKs).
- Es una alternativa ideal para comunicaciones en tiempo real(audio, video).
- Cada datagrama UDP se mapea en un datagrama IP en nivel 3.
- La comunicación es simplex (unidireccional), permitiendo asimismo tráfico multicast o broadcast.
- No mantiene el estado de la conexión.
- No asegura la ordenación de los datos, lo que conlleva a que se haya tenido que realizar la implementación de un número de secuencia en la cabecera de Network Coding.
- Si un datagrama IP o UDP se pierde será problema de la aplicación correspondiente incorporar mecanismos de retransmisión.

- El checksum es opcional, aunque generalmente se utiliza. La comprobación de este parámetro permite descartar datagramas erróneos y no genera ningún tipo de mensaje hacia el origen. La cabecera UDP incluye un campo de 16 bits, calculado por el transmisor a partir de las cabeceras IP, UDP y de los datos.
- No suministra ningún mecanimo de control de flujo o congestión; por tanto. si se va a realizar el envío de múltiples mensajes, y bajo una situación de congestión de la red o saturación del receptor, los datagramas serán descartados silenciosamente, esto es sin informar al emisor ni al receptor. No obstante, en algunos casos se contemplan mecanimos en el nivel de aplicación, que permiten que el receptor detecte si se produce alguna pérdida.

### 2.4.1. Sequence Number

UDP es un protocolo de transporte muy sencillo que, como ya se ha comentado, no incorpora en su cabecera ningún mecanismo de control de flujo como puede ser el número de secuencia insertado en cabecera TCP, lo que en este trabajo es una necesidad, ya que el receptor necesitará conocer el orden de los datagramas recibidos para reconstruir la información final. Esta funcionalidad ha sido incorporada en la entidad introducida entre el nivel de tranpsorte y de red, Network Coding, tal y como se muestra en la Figura 2.7.

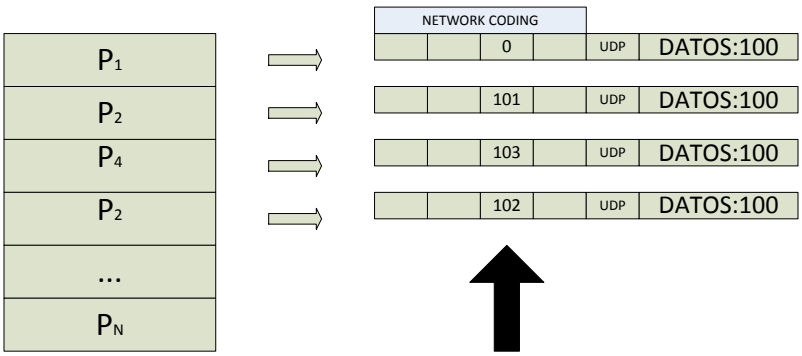



Figura 2.7: Implementación del Sequence Number

## 2.5 Antecedentes

Desde su creación en el año 2000, Network Coding se ha previsto como una técnica alternativa al esquema tradicional *store and forward*. Ahlswede [2] introdujo el concepto inicialmente. Básicamente se trata que los nodos intermedios de la red tienen la capacidad de codificar paquetes, produciendose una serie de ventajas como la mejora del rendimiento (aumento del throughput, mayor robustez, reducción del consumo energético) o el incremento de la seguridad, protegiendo las comunicaciones inalámbricas contra posibes ataques.

Por ejemplo, una de las aportaciones más populares que ha ayudado a NC a ganar su relevancia actual, fue el trabajo realizado por Katti [3], que introdujo el análisis del rendimiento TCP y UDP sobre redes inalámbricas. Fue el primero en hacer uso de Network Coding sobre estas redes, llegando a realizar simulaciones con redes hasta veinte nodos. La nueva arquitectura de envío, denominada COPE, cuenta con interesantes características, como el envío periódico de *reception reports* para indicar a los vecinos de un nodo qué paquetes se han almacenado en su Decoding Buffer (hace uso de piggybacking). Sin embargo, bajo situaciones de congestión de tráfico de la red estos informes pueden perderse, mientras que en supuestos de poco tráfico, podrían llegar demasiado tarde, después de que el nodo codificador haya tomado una decisión. Un nodo no puede basarse en los *reception reports*, por lo que se calcula una probabilidad de envío entre cada enlace, asignando diferentes pesos a los nodos; estos se tendrán en cuenta a la hora de codificar el paquete, utilizando la métrica ETX [4]. COPE incluye un sistema de envio de ACKs asíncronos, para solventar el problema de confirmación de paquetes de 802.11. COPE mejora significativamente el throughput de flujos UDP hasta en 3/4 veces, mientras que para los flujos TCP solo se observaron mejoras del 2 % - 3 %. COPE nunca ha sido presentado como un sistema totalmente efectivo.

No se han realizado muchos más trabajos estudiando la combinación de técnicas Network Coding, los patrones de tráfico UDP y las redes inalámbricas; no obstante se podría nombrar alguno en combinación con TCP como en [5] donde se propone un protocolo similar a COPE, en el que los nodos codifican en función de lo que han enviado y recibido anteriormente, por lo que no se realiza el proceso de *oportunistic listening*. Los paquetes llevan asociado un temporizador, cuyo aumento, conlleva un incremento de las oportunidades de codificación. El estudio demuestra que un temporizador óptimo puede llevar a incrementos de throughput de entre un 20 % y un 70 %.

Se puede también encontrar algún estudio similar a este trabajo como [6], en él se han realizado simulaciones de los escenarios con topología X y *Butterfly* con TCP pero, en el que se sustituyen los enlaces inalámbricos por el medio cableado, al que se le añaden errores aleatorios. Este trabajo pone de manifiesto las virtudes de Network Coding ya mencionadas, como el incremento del throughput, pero únicamente hasta un cierto nivel de error, a partir del cual el esquema de envío tradicional se vuelve más efectivo.

En cuanto a las técnicas de codificación, en este trabajo se realiza una operación sencilla XOR pero existen estudios sobre técnicas más avanzadas y complejas, como en [7], que investiga técnicas de codificación más avanzadas o [8], que propone combinaciones aleatorias más complejas. Desde otro punto de vista, hay que tener en cuenta el impacto del uso de estas técnicas más complejas, ya que harán que aumente el tiempo de procesado del nodo, lo que puede verse como una cancelación de los beneficios obtenidos.

Hasta el momento, se ha estado hablando de simulaciones y modelos teóricos. En [9] y [10] se presentó la primera implementación real de *Network Coding*. Se trata de un prototipo de Microsoft que permite la distribución de contenidos en redes peer-to-peer(P2P), que se denominó *Avalanche*. Utilizan las técnicas estudiadas en este trabajo

con la finalidad de reducir los tiempos de transferencia de archivos. Respecto a otras aplicaciones que utilizan P2P sin técnicas NC, como BitTorrent, los estudios muestran que el uso de dicha técnica reduce entre un 20 % y un 30 % el retardo. En este tipo de redes, el servidor no transfiere el archivo a cada usuario individualmente, sino que lo divide en varios fragmentos. *Avalanche* codifica aleatoriamente varios de estos fragmentos, que son enviados a los usuarios y éstos, a su vez, comparten estos trozos con el resto de usuarios. Los intercambios de los fragmentos codificados sólo se puede producir entre usuarios vecinos.

# 3
# Network Simulator

En este capítulo de la memoria se va a introducir la principal herramienta utilizada para la realización del trabajo. Se abordará la estructura interna de alguno de sus elementos; aquellos que, por necesidad, han sido más utilizados y, por último, se realizará una primera aproximación a los cambios añadidos en varios de estos módulos.

## 3.1 Características

En 1989 aparece un simulador de redes, desarrollado por Srinivasan Keshav conocido como REAL, y que es el antecesor a NS. La primera versión del NS se remonta al periodo 1995-1997, y fue desarrollada en el Laboratorio Nacional Lawrence Berkeley. Su núcleo fue escrito en C++, con scripts Tcl. McCanne reemplaza el uso de Tcl por OTcl (Tcl orientado a objetos) en 1996-1997, denominándose esta nueva versión del simulador como NS-2. En 2006 se comienza a desarrollar una nueva versión, NS-3, que ha sido la que se ha utilizado en este trabajo. Es un simulador novedoso porque ha sido escrito desde cero, siendo incompatible con NS-2 a pesar de que sigue empleando C++. También funciona con módulos programados en Phyton. NS-3.1 fue lanzado en 2008, y ha ido recibiendo actualizaciones de manera continua, siendo NS-3.18 la vesión más reciente, de agosto de 2013.

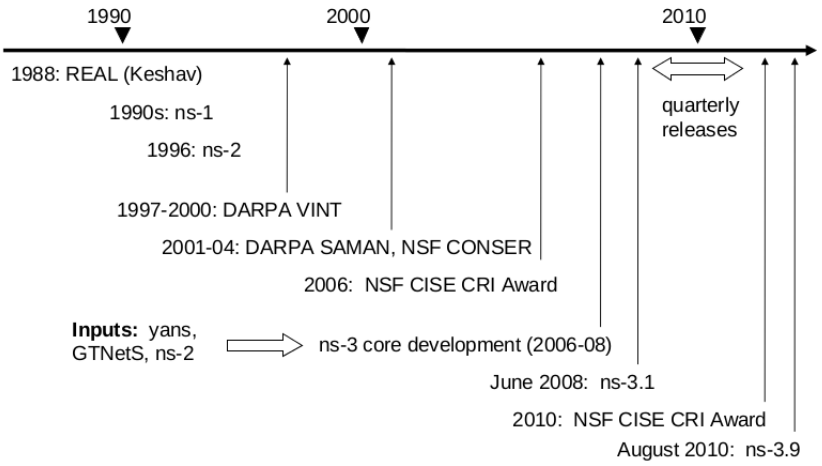



Figura 3.1: Cronología *Network Simulator*

No se han usado las versiones más recientes de NS, debido a la inestabilidad que

podrían acarrear y a que, además, se han utilizado ciertos módulos desarrollados para la versión 3.13 de diciembre de 2011. De todas formas, las mejoras que ofrecen las actualizaciones posteriores son escasas, centrándose en la solución de pequeños errores.

Una característica a destacar es que se trata de software libre, bajo licencia GNU GPLv2, y está disponible al público, para la investigación y el desarrollo, entornos académicos, la industria o incluso gobiernos. Cabe destacar su flexibilidad y su jerarquía modular, que brinda la posibilidad de incorporar y/o modificar módulos con funcionalidades que no habían sido implementadas, y que abre un amplio abanico de opciones de simulación, permitiendo realizar estudios a gran escala, además que hoy en día son absolutamente imprescindibles. Para ello es necesario tener un conocimiento profundo de la arquitectura del simulador.

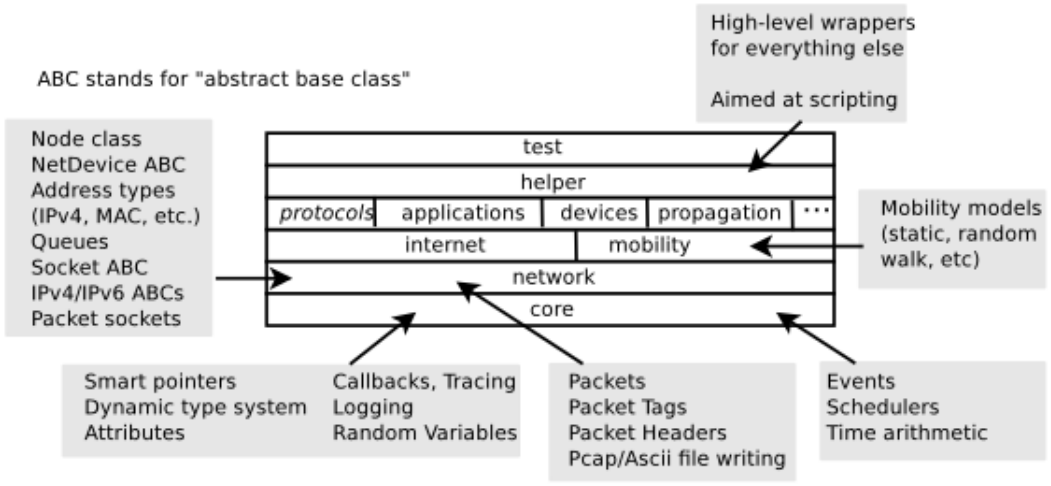



Figura 3.2: Organización software de NS-3

NS-3 es un simulador de redes de eventos discretos [11], donde el núcleo y todos los módulos están implementados en C++. Sintetizando, se parte de un programa principal, donde se define la topología de la simulación, las caracteŕisticas básicas (por ejemplo, el tamaño de los paquetes, el tiempo de la conexión) y es donde se enlaza con las diferentes librerías de NS-3. Además, NS-3 tiene exportados la mayoría de sus API a Phyton, permitiendo crear escenarios de simulación también en este lenguaje.

El código fuente de NS-3 se encuentra mayormente organizado en el directorio /src y su estructura es la que se muestra en la Figura 3.2. Generalmente las dependencias de los módulos se limitan a las que están en el nivel inmediatamente superior. Es importante, a la hora de hacer uso de esta herramienta, tener claro la estructura de cada uno de los módulos, ya que facilitará su control posteriormente.

La financiación inicial para desarrollar NS-3 fue proporcionada por la fundación National Science y el grupo Planete en INRIA Sophia Antipolis. Posteriormente se ha contado con la ayuda del Departameto de Ingeniería Eléctrica y Computación en el Instituto de Tecnología de Georgia, y el Departamento de Ingeniería eléctrica de la Universidad

de Washington. El grupo de investigación de Redes Inalámbricas en INESC Porto, de la Universidad de Oporto, tambíen ha apoyado el proyecto desde sus inicios. El Google Summer of Code ha incluído NS-3 como uno de los participantes en 2008, 2009 y 2010.

## 3.2 Componentes

NS3 está compuesto por una gran multitud de elementos que, a su vez, están formados por varios ficheros de código C++, en los que se declaran las diversas clases, sus dependecias y funciones. Por tanto, a continuación, se explicarán los módulos más relevantes en relación con el trabajo que se presenta en esta memoria.

### 3.2.1. Aplicación

A nivel de aplicación, se han utilizado dos implementaciones: cliente y receptor.

- OnOffApplication: aplicación que se implementa en el transmisor, generando un tráfico estable hacia un único destino. Permite configurar una serie de parámetros que determinan el patrón de tráfico a generar. Los más importantes son: Ontime, Offtime, DateRate, PacketSize y MaxBytes. El primero es una variable que indica la duración del estado de transmisión (modo ON), mientras que el segundo refleja el tiempo de parada (modo OFF). DateRate parametriza la velocidad de la fuente, PacketSize se refiere al tamaño del paquete, y MaxBytes es el número total de bytes enviados por la aplicación.

- PacketSink: aplicación que se implementa en el nodo receptor y va unida, como la pareja, a OnOffApplication. Recibe tráfico según una dirección y puerto destinos. Sus parámetros de funcionamiento son *Local*, que asigna la dirección donde van a recibirse los datagramas, *Protocol* que define el identificador en el socket receptor (en este trabajo UDP). En esta clase se definen los métodos *StartApplication* y *StopApplication*, que definen el intervalo temporal en el que la aplicación está preparada para recibir paquetes.

Las aplicaciones mencionadas han sido empleadas para generar el tráfico empleado en este trabajo a través de las simulaciones de las diversas topologías que serán comentadas en el capítulo 5.

### 3.2.2. UDP

Gran parte del trabajo llevado a cabo gira en torno al recorrido de un paquete a través de la pila de protocolos UDP/IP. En este proyecto se hace uso de la implementación nativa de UDP para NS-3 que, a nivel interno, se organiza de la siguiente manera:

- UdpSocket class: establecimiento de los atributos del socket, que pueden ser usados a lo largo de las diversas implementaciones. Se encuentra definida en los ficheros */src/internet/model/udp-socket(.cc/.h).*

- UdpSocketFactory class: se encarga de crear sockets UDP. Define una interfaz de programación para los Sockets (API) y almacena las variables globales usadas por defecto a la hora de inicializar un socket. Se encuentra definida en los ficheros */src/internet/model/udp-socket-factory(.cc/.h)*
- UdpL4Protocol class: clase que define una capa intermedia entre el socket y el nivel 3, IP, que facilita la interconexión entre estas capas, gracias a la asignación del *end point*, que se encarga de la correspondencia(*IpV4EndPoint*). Se encuentra definida en los ficheros */src/internet/model/udp-l4-protocol(.cc/.h)*
- UdpSocketImpl class: definición de los aspectos específicos del socket. Se encuentra definida en los ficheros */src/internet/model/udp-socket-impl(.cc/.h)*
- IpV4EndPoint class: almacenamiento del puerto local, dirección local, puerto destino, y dirección destino asociados al socket. Se encuentra definida en los ficheros */src/internet/model/ipv4-end-point(.cc/.h)*
- UdpHeader class: implementación de los campos que constituyen la cabecera UDP (número de puerto, datos,etc). Se encuentra definida en los ficheros */src/internet/model/udp-header(.cc/.h)*
- UdpSocketFactoryImpl: se trata de la implementación de UdpSocketFactory para UDP nativo. Se encuentra definida en los ficheros */src/internet/model/udp-socket-factory-impl(.cc/.h)*

Muchos de los objetos internos, o detalles de implementación, están expuestos en la API pública, que facilita la creación de implementaciones alternativas a los modelos nativos. La API pública de esos objetos en C++ se encuetra en el directorio */src/network/model*, a destacar principalmente: *adress.h*, *socket.h*, *node.h* y *packet.h*.

En la Figura 3.3 se representa el recorrido de un paquete en los nodos que tienen implementada la pila de protocolos UDP. Hay que tener en cuenta que es independiente de las capas físicas y de enlace.

El diagrama de la izquierda muestra el flujo que sigue un paquete en transmisión. A nivel de aplicación se llama al socket para enviar el paquete, que está definido por la clase *UdpSocketImpl*. El socket se apoya sobre *UdpL4Protocol*, para dirigirse al destino (en UDP no hay un establecimiento de la conexión). Crea el datagrama UDP, añadiendo la cabecera correspondiente, y lo envía al nivel inferior *Ipv4L3Protocol*. En este módulo se encuentra definida la capa IP, que añadirá su cabecera y enviará el paquete a *NetDevice*. En la transmisión hay un paso previo, el protocolo Address Resolution Protocol(ARP), que se recoge en el módulo *ArpIpv4Interface*, que averigua la dirección MAC del nodo al que desea transmitir.

La figura de la derecha representa el proceso contrario: cuando un paquete es recibido en el nodo hasta que llega al nivel de aplicación. Cuando *NetDevice* recibe el mensaje, llama a la función receiveCallback, que va a detectar el protocolo con el que se está trabajando. En este caso, *Ipv4L3Protocol*, que se encargará de eliminar la cabecera

IP, así como de redirigir el paquete al nivel UDP, *UdpL4Protocol*, donde se eliminará la cabecera UDP, asociándose al *Ipv4EndPoint* con la dirección-puerto destino. Finalmente, se encaminará el paquete hacia el socket correspondiente, llegando así al nivel de aplicación.

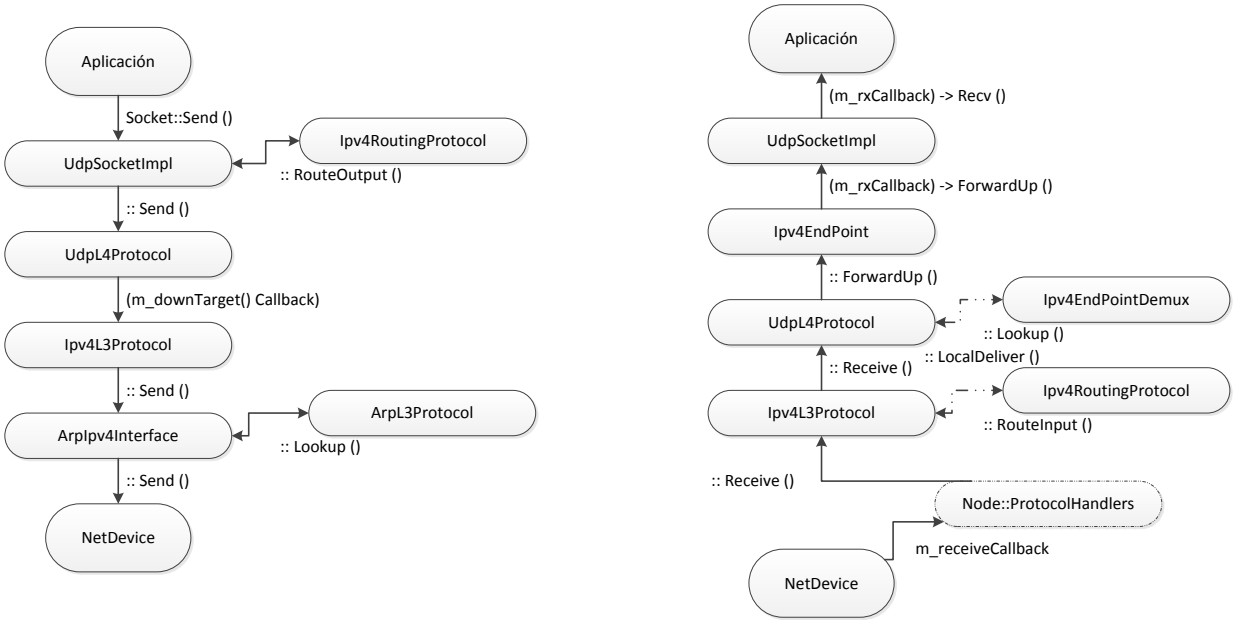



Figura 3.3: Flujo de un paquete entre los objetos (capas) del simulador NS-3

## 3.3 Network Coding en NS-3

El simulador no cuenta con ninguna implementación nativa de *Network Coding* y es, por tanto, necesario hacer uso de la versatibilidad comentada para añadir un módulo que implemente dicha técnica.

El módulo implementado se encuentra en */src/network-coding* y lo componen los siguientes ficheros, cuyo desarrollo será explicado en el siguiente capítulo.

- *NetworkCodingBuffer* class: recoge las estructuras y funciones necesarias para el funcionamiento de los buffer de codificación, decodificación, tablas hash. Se encuentra en*/src/network-coding/model/network-coding-buffer(.cc/.h)*
- *NetworkCodingHeader* class: implementación de la cabecera de *Network Coding*, que se situa entre transporte y red. Se encuentra en */src/network-coding/model/network-coding-header(.cc/.h)*
- *NetworkCodingL4Protocol* class: nuevo nivel en la pila de protocolos, que se encarga de las funciones desempeñadas por tal entidad, así como la interconexión entre las

capas UDP y NC. Se encuentra en */src/network-coding/model/network-coding-l4-protocol(.cc/.h)*

- *UdpNetworkCoding* class: consta de las principales funciones desempeñadas por los nodos que componen el escenario. Se encuentra en */src/network-coding/model/UDP-simple-network-coding(.cc/.h)* Se pueden destacar las siguientes funciones:

  - *Send*: los nodos transmisores envían el datagrama. En esta función se crea la cabecera de *Network Coding*. Por tanto, despues de que *OnOfApplication* genera los datos (1437bytes), esta función la encargada de añadir el número de secuencia correspondiente a cada datagrama, incorporándolo a la cabecera NC y adjuntar ésta al paquete.
  - *ParseForwardingReception*: se encarga de almacenar los paquetes nativos en el *Coding Buffer*; si se trata de un paquete codificado éste será retransmitido (*store-and-forward*).
  - *SendDown*: función clave en el *Coding Node*; en un nodo intermedio un paquete normalmente alcanza únicamente el nivel IP, pero en este caso, es necesario llegar a NC para poder realizar la posterior codificación. La función se encarga en particular de la modificación explícita de la cabecera de NC, como es añadir la información de cada paquete nativo en la misma.
  - *Receive*: función utilizada por los nodos receptores cuando no hay codificación ( o si NC está desactivado), que envía el datagrama al nivel UDP.
  - *ReceivePromiscuous*: función que realiza el *overhearing*, así como el proceso de decodificación. Si llega un paquete nativo será almacenado en el *Decoding Buffer*, mientras que si es un paquete codificado se procesará la cabecera NC, buscando en dicho buffer si alguno de los paquetes nativos guardados coincide con los que están en el codificado, de forma que pueda producirse una decodificación satisfactoria.
  - *ForwardUp*: su finalidad es enviar los datos procesados a la capa superior (protocolo UDP), incluyendo el paquete nativo y la correspondiente cabecera UDP.

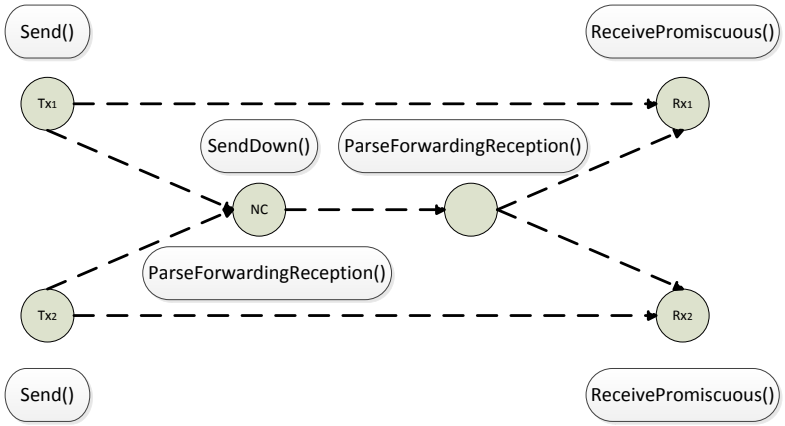



Figura 3.4: Implementación de las principales funciones de NC en los nodos de la red

# 4
# Desarrollos en el marco de NS3

## 4.1 Network Coding en NS3

En las siguientes páginas están descritos todos los módulos de los que se compone la implementación de las técnicas de Network Coding en NS3. Se han incluido en */src/network-coding*. Los procesos correspondientes han sido programados en C++, y no se explicará ningún elemento a nivel de código, para agilizar la lectura de la memoria.

### 4.1.1. Cabecera Network Coding

Considérese que se tiene un paquete codificado, que alcanza uno de los posibles nodos receptores para uno de los paquetes nativos que lo forman. La cuestión es ¿cómo puede reconocer el destino si uno de ellos está dirigido a él?. En principio no puede saberlo, ya que toda la información útil viaja codificada en el paquete y tampoco puede realizarse su decodificación, porque no se dispone de toda la información necesaria. Por tanto, surge la necesidad de que el nodo receptor disponga de tales datos, para lo que se incorpora una cabecera de NC que permite que un nodo sepa como debe procesar un paquete codificado, ya sea reenviar, descartar o decodificar.

Siempre va a estar formada por tres campos fijos: protocolo, tipo y número de paquetes codificados. Este último parámetro determinará la longitud variable del siguiente bloque, formado por ya que un grupo de elementos por cada paquete codificado: IP destino, IP origen, número de secuencia, código hash y longitud de los datos.

A continuación se presenta la cabecera NC, junto a una breve explicación de los campos que la componen.

- Protocolo. Campo que indica el protocolo de transporte sobre el que se sustenta el envío de datos. En este caso, UDP, esta identificado por el número 17.
- Tipo: indica si el datagrama es de datos, mediante un cero.
- Número de paquetes codificados: parámetro que define el número de paquetes nativos que están recogidos en el codificado. Toma un valor mayor o igual que uno, ya que por cada paquete se necesita toda la información necesaria para que pueda llegar a su destino.

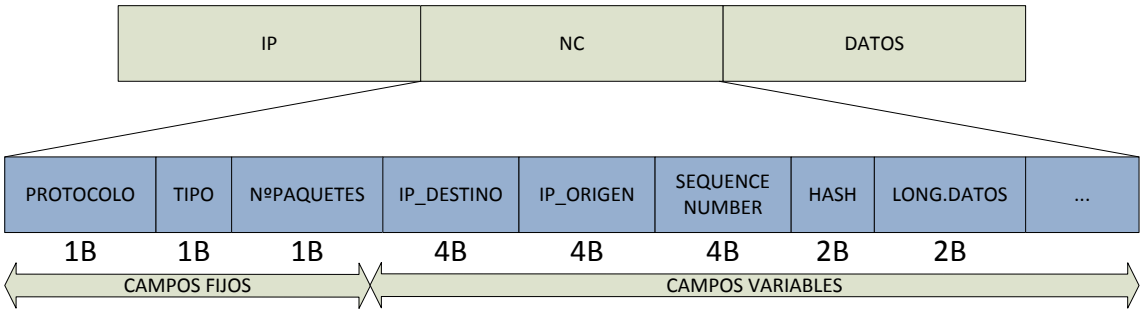



Figura 4.1: Cabecera correspondiente al nuevo nivel Network Coding

- IP destino: dirección del nodo receptor a nivel 3. El destino tendrá que consultar en esta cabecera si está interesado en este paquete o no. Como se comentó previamente, esto no es un problema, por la propiedad de *overhearing* del receptor. Está implementado para que en este campo se copie la IP del paquete que se registra primero en *InputPacketPool*.
- IP origen: dirección del emisor a nivel 3, necesaria para que el destino conozca la procedencia del paquete nativo.
- Número de secuencia: parámetro que facilita al receptor la reordenación de la información original. Como se ha comentado, UDP es un protocolo muy sencillo que no introduce este mecanismo, ya que tiene una operación no orientada a conexión. Pero en el ámbito de este proyecto es una funcionalidad que se ha tenido que implementar para que la información puede estar ordenada en el nivel de aplicación.
- hash: define un valor hexadecimal a traves de una operación *hash* con la IP origen, IP destino, Puerto origen, Puerto destino, y sirve para identificar un flujo de manera unívoca.
- Longitud de los datos: muestra la longitud del campo de datos de la entidad NC en bytes.

Como se puede observar, el tamaño de esta cabecera no es muy elevado, por lo que la sobrecarga que introduce a los datagramas es baja. Incluso en el caso de que se codifiquen un número importante de paquetes, las mejoras que introduce NC con su envío simultáneo supera con creces este inconveniente.

### 4.1.2. InputPacketPool y DecodingBuffer

Al dotar de cierto nivel de inteligencia al nodo intermedio, Coding Node, éste tiene que contar con un buffer donde se depositan los paquetes de los flujos de datos a la espera de una oportunidad de codificación. En este trabajo dicho buffer recibe el nombre de *InputPacketPool*, que se caracteriza por utilizar números hash para identificar los flujos. Cuando llega un paquete, se deposita inmediatamente si está vacio, mientras que si no lo

está se realiza una búsqueda para encontrar un datagrama de un flujo distinto apareciendo así una oportunidad de codificación, en caso contrario, el paquete será almacenado. Este buffer lo caracterizan parámetros de los que se ha hablado anteriormente: Buffer Size y Buffer TimeOut.

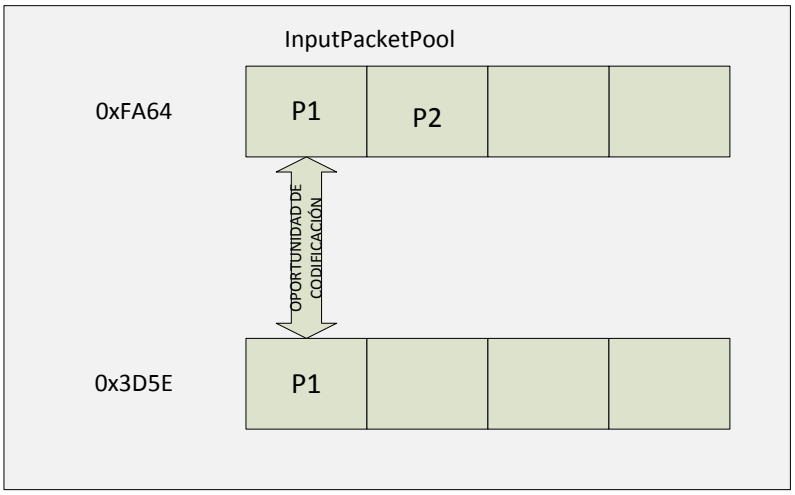



Figura 4.2: Input Packet Pool

La necesidad de almacenar los paquetes nativos para las posteriores labores de decodificación hace que la naturaleza broadcast de las redes inalámbricas, las convierta el medio ideal, ya que permite que todos los nodos escuchen y reciban paquetes que no van dirigidos hacia ellos, pero que se transmiten dentro de su área de cobertura. Este proceso recibe el nombre de *overhearing*. Son transmisiones unicast que, por la naturaleza del medio de propagación pueden llegar a destinos no previstos. Es el nivel de enlace, 802.11, el que se encarga de comprobar la dirección MAC de destino de los paquetes, descartándolos si no van dirigidos al nodo receptor. Por tanto, aquí es donde habrá que realizar la modificación pertinente para permitir la implementación de buffers que se encargarán de almacenar posibles paquetes nativos.

Por tanto, resulta imprescindible contar en cada uno de los nodos receptores con un *Decoding Buffer*. Al estar trabajando en un entorno inalámbrico, el nodo realizará el overhearing de todos los paquetes que lleguen hacia él. La consecuencia es que los buffer se podrían llenar de información innecesaria, por lo que sería necesario realizar un filtrado para que solo puedan guardarse datagramas UDP que no vayan dirigidos expresamente a ese nodo.

### 4.1.3. Proceso de codificación - Nueva entidad entre IP y UDP

La fase de codificación se lleva a cabo por la entidad NC en el *Coding Node*. Utilizando como referencia un modelo de capas convencional, se encontrará situada entre el nivel 4, transporte, y el nivel 3, red.

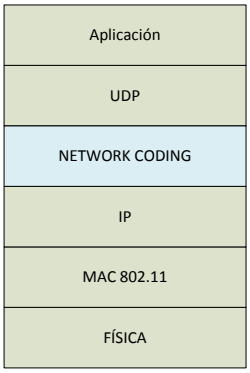



Figura 4.3: Pila de protocolos con Network Coding añadido

NC se situa por encima de IP, porque en la codificación se extrae información útil para la creación de la cabecera, así como su posterior decodificación. A lo largo de estás páginas se explicará el proceso más detalladamente. De esta forma los niveles inferiores actúan con normalidad hasta alcanzar la nueva capa implementada en NS3.

Una vez se alcanza el nuevo nivel, se ejecutan una serie de pasos que quedan representados en la Figura 4.4, donde se ve que la misión del Coding Node es la modificación de la nueva cabecera, añadida al resultado de la operación XOR, así como su posterior reenvío.

A continuación se enumeran las fases más relevantes del proceso.

1. Cuando un paquete UDP alcanza el *Coding Node*, la primera labor a llevar a cabo es comprobar si se puede tratar como paquete nativo. Para ello el número de paquetes codificados que lleva en la cabecera de NC debe de ser uno. En caso contrario el nodo procederá a su reenvío, para que continue su recorrido por la red.

2. Es el instante en el que se emplea el buffer *InputPacketPool*. Si está vacio el paquete se deposita automáticamente, identificado por su flujo mediante un código hash (IPdestino, IPFuente, PuertoDestino, PuertoFuente), de forma que el próximo paquete que se reciba comparará su hash con los guardados en el buffer. Si son distintos, aparece una oportunidad de codificación, y en caso contrario se añadirá al buffer junto a su flujo correspondiente. Por supuesto, hay que destacar el *Buffer Size* y *Buffer Time*, dos parámetros cuyo incremento aumentará las oportunidades de codificación. El primero se refiere al tamaño, mientras que el segundo se trata de un temporizador, que, al expirar, hace que el paquete que tiene asociado sea transmitido a su siguiente destino, dejando un espacio libre en el *Coding Buffer*.

3. Oportunidad de codificación: se realiza la operación XOR entre ambos paquetes, tanto los datos útiles como su cabecera UDP.

4. El siguiente paso consiste en construir la cabecera NC del nuevo paquete codificado, cuyas modificaciones más relevantes son:

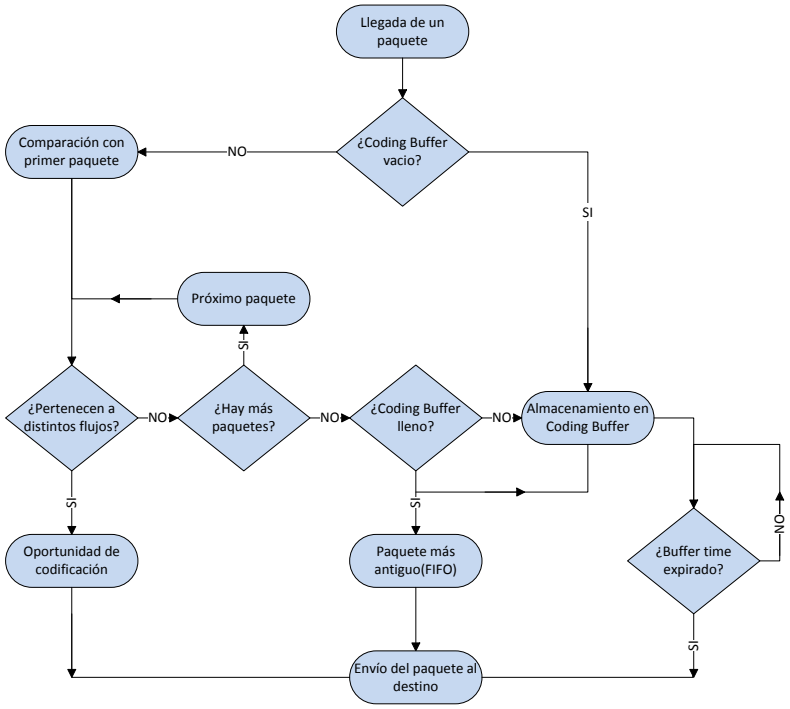



Figura 4.4: Flujo que recorre un paquete cuando alcanza el nodo codificador

- CodedPackets: número de paquetes nativos recogidos en el codificado.
- Por cada paquete nativo se introduce las direcciones IP del nodo destino y origen, obtenidos de la cabecera IP, el número de secuencia, que se ha tenido que implementar porque su uso es poco habitual en un protocolo no orientado a conexión, el hash y la longitud de los datos.

5. Se añade la cabecera NC y se envía a la red, estableciendo como dirección IP destino la de uno de los paquetes codificados. Los nodos destino deberán recibir el paquete por overhearing, aunque no esté dirigido a ellos.

6. Nótese que el *Coding Node* se encarga de eliminar del buffer los paquetes nativos junto a sus respectivos temporizadores.

### 4.1.4. Proceso de decodificación

El *Decoding Buffer* es un elemento central en todo este proceso, ya que va a ser el encargado de guardar los paquetes nativos no útiles, pero necesarios para que el receptor pueda recibir su paquete original. Obsérvense los pasos a seguir durante este proceso en la Figura 4.5.

1. Tras la recepción de un paquete en la capa de NC se comprueba la cabecera correspondiente, en concreto el campo de número de paquetes codificados, que indica si se trata de uno nativo o no. Si su valor es uno, es efectivamente un datagrama UDP

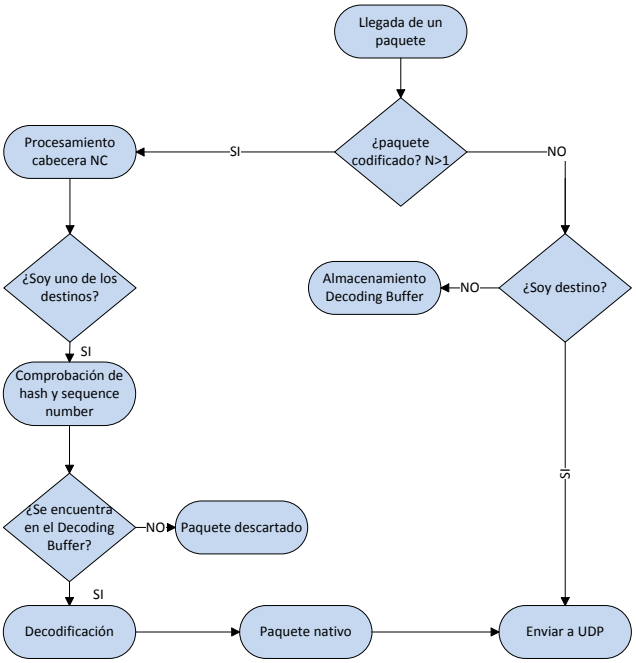



Figura 4.5: Flujo que recorre un paquete cuando alcanza el nodo receptor

nativo que si no va dirigido al receptor será almacenado en el *Decoding Buffer*. Sin embargo, si es distinto de uno, se sabe que se trata de un paquete codificado.

2. Una vez que se conoce el número de paquetes nativos de los que consta el codificado, se procede a analizar el resto de campos, ya que, como se ha mencionado antes, se dispone de un grupo de elementos por cada paquete nativo codificado. El nodo comprobará si alguno de los destinos de los paquetes codificados va dirigido a él, y si es así, se procederá a la decodificación.

3. Comienza el proceso de decodificación. Se busca en el *Decoding Buffer* si se encuentra almacenado algún paquete con el mismo flujo y número de secuencia de alguno de los paquetes que constituyen el codificado. Los escenarios analizados tienen únicamente dos flujos, y el número de secuencia será único, por tanto no hay que tomar mayores medidas para saber si se trata del datagrama requerido. A continuación se realiza la operacion XOR entre el paquete nativo y el codificado.

4. Si la decodificación se completó correctamente, el siguiente paso consiste en formar la cabecera IP con la información de la cabecera NC. fiinalmente, el paquete nativo será enviado a la capa superior, UDP.

## 4.2 Archivos de configuración del canal

A pesar de que la principal aportación de este trabajo residen en la implementación de Network Coding, se han desarrollado además otras clases con la finalidad de facilitar

el proceso de medidas. En primer lugar, se destaca la clase *ConfigureScenario*, que se va a encargar de poner en marcha las bases de la simulación a traves de cuatro ficheros de texto: *network-coding-scenario.conf*, *x-channel-total.conf*, *x-scenario.conf* y *x-static-routing.conf*.

Todos los ficheros descritos en esta sección se han particularizado para los tres escenarios que se emplearán durante el proceso de medidas, usando la X como ejemplo ilustrativo.

### 4.2.1. *network-coding-scenario.conf*

Se trata de un archivo de texto que es utilizado constantemente, pues se encarga de establecer todos los parámetros principales de la configuración del escenario y de la simulación, como son el número de simulaciones, el número de paquetes transmitidos, la longitud de los datos, FER, protocolo de transporte, la configuración de la funcionalidad Network Coding, los archivos traza de salida, etc.

```
[SCENARIO]
RUN=100
RUN_OFFSET=0
NUM_PACKETS=10000
PACKET_LENGTH=1437
FER=0.0

[STACK]
TRANSPORT_PROTOCOL=UDP
ROUTING_PROTOCOL=STATIC
PROPAGATION_LOSS_MODEL=MANUAL
NODE_DEPLOYMENT=FILE
SCENARIO_DESCRIPTION=x-scenario.conf
CHANNEL_CONFIGURATION=x-channel-sides.conf
STATIC_ROUTING_TABLE=x-static-routing.conf

[NETWORK_CODING]
ENABLED=1
PROTOCOL=UdpNetworkCoding
BUFFER_SIZE=10
BUFFER_TIMEOUT=100
MAX_CODED_PACKETS=2
```

Figura 4.6: Fichero de configuración inicial

### 4.2.2. *x-scenario.conf*

Se define la posición de los nodos, a través de su identificador, su punto de coordenadas(x,y,z), nodos que transmiten(TX), nodos que toman el papel de receptores (RX) o si sólo participan en el procesode reenvio paquetes (FWD). En la Figura 4.7 se muestra la configuración utilizada para la topología de la X.

El resultado de la configuración anterior tiene como resultado la topología de la Figura 4.8:

| #No. | X | Y | Z | TX | RX | CR | FWD |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 5 | 0 | 0 | 0 |
| 2 | 0 | 25 | 0 | 4 | 0 | 0 | 0 |
| 3 | 7.5 | 12.5 | 0 | 0 | 0 | 1 | 1 |
| 4 | 15 | 0 | 0 | 0 | 1 | 0 | 0 |
| 5 | 15 | 25 | 0 | 0 | 1 | 0 | 0 |

Figura 4.7: Fichero de configuración espacial de los nodos

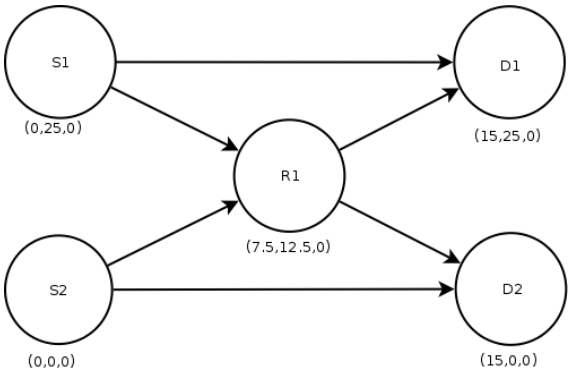



Figura 4.8: Representación física del fichero

### 4.2.3. *x-channel-total.conf*

Para la configuración del canal se hace uso de este fichero, que tiene formato de matriz, cuyos valores toman el siguiente significado:

- 0 - Ideal. No hay errores en este enlace, todos los paquetes que se envían llegan al destino (FER=0).
- 5 - Hay errores, la FER en este caso es la que corresponde a la indicada por el usuario en el archivo de configuración inicial: *network-coding-scenario.conf*
- 1 - No hay enlace entre este par de nodos.

La Figura 4.9 muestra la configuración *normal* para el escenario de la Figura anterior. En este caso todos los enlaces tienen la misma FER; por tanto sería el caso más natural y realista convirtiéndose en el que puede proporcionar una idea más precisa acerca del verdadero rendimiento de NC.

| | | | | |
|---|---|---|---|---|
| 1 | 5 | 5 | 5 | 5 |
| 5 | 1 | 5 | 5 | 5 |
| 5 | 5 | 1 | 5 | 5 |
| 5 | 5 | 5 | 1 | 5 |
| 5 | 5 | 5 | 5 | 1 |

Figura 4.9: Fichero de configuración de canal

Se han trabajado con otras dos configuraciones respecto a la FER.

1. *Sides* - Esta configuración proporciona un canal libre de errores en todos los enlaces excepto en aquellos en los que se realizará el proceso de *overhearing*, es decir en los de los extremos. De esta forma es posible aislar y estudiar cuál es la influencia de este tipo de pérdidas respecto al rendimiento de NC.

2. *Middle* - Esta distribución introduce errores únicamente en el enlace central, una vez que la codificación ya ha sido realizada, cuya finalidad es conocer cual es la influencia que produce sobre el rendimiento UDP la pérdida de paquetes codificados.

### 4.2.4. *x-static-routing.conf*

Se realiza un procedimiento muy parecido a los anteriores para acometer la configuración de encaminamiento de los nodos. Hay que indicar el destino, el próximo salto e interfaz, y indicando además todas las posibles rutas. Un ejemplo de este fichero se recoge en la Figura 4.10.

```
#Node ID        Destination     Nexthop         Interface
0               1               2               1
0               2               2               1
0               3               3               1
0               4               2               1
1               0               2               1
```

Figura 4.10: Fichero de configuración de encaminamiento estático

## 4.3 Ficheros traza de salida

En el proceso de caracterización, surge la necesidad de plasmar, cuantificar los resultados de las trazas a través de gráficos, para facilitar su comprensión. Aunque existen varias formas de obtener datos que son objeto de estudio, se ha optado por desarrollar una clase nueva, *PropietaryTracing*, donde se recogen todas las transmisiones realizadas que, posteriormente, serán filtradas, gracias a un programa de cálculo como Matlab. Esta traza se la denomina, de manera informal, traza larga, porque este primer formato sirve para realizar un estudio temporal del escenario, incluyendo todos los datagramas recibidos por cada nodo, permitiendo localizar posibles errores y fallos. Los campos que se recogen para este estudio son:

Tabla 4.1: Formato fichero de traza larga

| Time | Tx/Rx | Nodo ID | Source IP | Dest. IP | Src Port | Dst Port | Length | SeqNum | Coded pkts | Decoded |
|---|---|---|---|---|---|---|---|---|---|---|
| 20.020 | 2 | 0 | 10.0.0.1 | 10.0.0.5 | 49157 | 50000 | 1437 | 27285 | 1 | 0 |
| 20.021 | 0 | 4 | 10.0.0.1 | 10.0.0.5 | 49157 | 50000 | 1437 | 4309 | 2 | 1 |

- *Time*: representa el instante temporal en que se ha recibido el datagrama.
- *Tx/Rx*. Parámetro que permite establecer el proceso en el que se encuentra el nodo y que puede tomar los siguientes valores.

a. Reception(Rx node): 0
b. Reception(*overhearing*): 1
c. Transmission(Tx node): 2
d. Transmission (*Coding Node*): 3

- *Node ID*: nodo que está actuando de Tx/Rx.
- *Source IP*: dirección IP que aparece en el campo fuente de la cabecera IP. Facilita conocer el nodo que ha generado el mensaje.
- *Dest. IP*: dirección IP que aparece en el campo destino de la cabecera IP, determina el nodo al que iba dirigido el mensaje.
- *Src Port*: indica el puerto origen de la cabecera UDP.
- *Dst Port*:indica el puerto destino de la caberca UDP.
- *Length*: tamaño de los datos, en bytes, que se están enviando en este datagrama.
- *SeqNum*: número de secuencia de la cabecera de Network Coding. En caso de ser un paquete codificado, inserta el del primer paquete de la cabecera.
- *Coded pkts*: toma valor uno si se trata de un paquete codificado.
- *Decoded*: toma valor uno si la decodificación ha sido satisfactoria.

Esto es idóneo cuando se están realizando de funcionamiento de la implementación, pero cuando se pretende hacer una simulación con valores más realistas, la cantidad de información que va a generar es inmensa y ralentiza todas las tareas. Además, no se necesita, para los posteriores cálculos una información tan detallada. Así que se decidió crear otra clase, *NetworkCodingHelper*, alojada en */src/network-coding/helper*, para aligerar el proceso, mostrando los resultados estadísticos que sí tienen mayor relevancia en cada simulación.

En la Tabla 4.2 se muestra el formato de esta traza resumen.

Tabla 4.2: Formato fichero de traza resumen

| No. | BufSiz | BufTo | MaxCP | FER | Coding Rate | Decod Rate | Throughput | Transmissions | Retardo | Jitter | Tx | Rx |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 | 4 | 20 | 2 | 0.10 | 0.7575 | 0.9062 | 1.4154 | 23843 | 0.0082 | 3.968e-05 | 20000 | 4991 |
| 22 | 6 | 20 | 2 | 0.10 | 0.7496 | 0.9123 | 1.4344 | 23885 | 0.0082 | 4.579e-05 | 20000 | 5021 |

Los principales parámetros que constituyen este fichero de salida son:

- *BufSiz*: indica el tamaño del buffer del *Coding Node*, que el usuario ha fijado en el archivo de configuracion principal *network-coding-scenario.conf*
- *BufTo*: fija el parametro del temporizador del *Coding Node*, que el usuario ha indicado en el archivo de configuracion principal *network-coding-scenario.conf*

- *MaxCP*: número de paquetes que han sido codificados.
- *FER*: parámetro que fija el grado de error en los enlaces del escenario analizado, según el fichero *network-coding-scenario.conf*
- *Coding Rate*: porcentaje de paquetes que han sido codificados.
- *Decod Rate*: porcentaje de paquetes que han sido decodificados con éxito
- *Throughput*: rendimiento de la transmisión. Parámetro básico en este trabajo, que muestra de manera directa los beneficios de la implementación de NC.
- *Transmissions*: número de transmisiones totales que se han realizado.
- *Retardo*: tiempo medio de la llegada de los paquetes al destino.
- *Jitter*: variación media del retardo entre recepciones consecutivas
- *Tx*: número de paquetes transmitidos totales.
- *Rx*: número de paquetes recibidos totales.

A partir de todos estos datos, se trabaja con ellos para llevar a cabo una serie de gráficos de las medidas más relevantes, de forma que se pueden analizar las ventajas y desventajas del sistema implementado.

## 4.4 Automatización de la simulación

Ahora bien, las simulaciones no pueden basarse en una sola medida para cada escenario. Hay que realizar un número mínimo para garantizar la validez estadística de los resultados, lo que supone un total de 18000 simulaciones. En concreto se ha decidido llevar a cabo cien simulaciones por configuración.

Para agilizar todo este proceso se ha empleado un script.sh (Figura 4.11) que utiliza un bucle continuo generando, todas las combinaciones posibles y que ejecuta, cada una de ellas, el fichero *network-coding-bash-script.cc*, alojado en la carpeta */src/scratch* cuyos parámetros de entrada se encuentran en el archivo *network-coding-scenario.conf*, que serán modificados por las variables de los bucles. Este archivo se va a encargar de leer el archivo de configuración, generando una estructura con todos los parámetros necesarios para ejecutar la simulación y va a ir completando el archivo traza resumen.

```
#! /bin/bash
for fer in  0.0 0.1 0.2 0.3 0.4
 do
   for timeout in 0 20 40 60 80 100
    do
      for buffersize in 0 2 4 6 8 10
       do
        for runoffset in 0 25
         do
           ./waf --run "scratch/network-coding-bash-script --Configuration=network-coding-scenario
             --RunOffset=$runoffset --Fer=$fer --Timeout=$timeout --BufferSize=$buffersize"
         done
       done
    done
done
```

Figura 4.11: script.sh encargado de automatizar la simulación

# 5 Simulaciones y resultados

En este capítulo se recogen, en primer lugar, una serie de conceptos que se consideran claves para comprender las medidas y resultados que se ilustrarán posteriormente. A continuación se presentan los parámetros con los que se han realizado las simulaciones y que ya se han nombrado en el capítulo anterior. En segundo lugar se explican los escenarios y las principales características de las tres topologías estudiadas a lo largo de todo el trabajo. Finalmente, se destacan los resultados más interesantes obtenidos al aplicar *Network Coding* con tráfico UDP sobre *Wireless Mesh Networks*

## 5.1 Conceptos fundamentales

Los conceptos claves que son necesarios para interpretar adecuadamente los resultados que se recogerán en la Sección 5.4 son los que aparecen a continuación.

- *Throughput*. Se define como el conciente entre los datos útiles correctamente recibidos en el equipo receptor y la duración total de la transmisión. Es uno de los principales parámetros de estudio en este proyecto, ya que aporta información básica acerca del rendimiento de la red.

$$Throughput = \frac{\text{Total bits de datos útiles}}{\text{Tiempo total de simulación}} \tag{5.1}$$

- *Coding Rate*. Se define como el cociente entre los paquetes codificados transmitidos y el total de paquetes enviados, a partir de la información disponible en el *Coding Node*. Este valor refleja el número de oportunidades de codificación que se han dado.

$$CodingRate = \frac{\text{Total paquetes codificados Tx}}{\text{Total paquetes Tx}} \tag{5.2}$$

- *Decoding Rate*. Se define como el cociente entre los paquetes decodificados y el número total de paquetes codificados recibidos, en los nodos receptores. Este valor refleja, el número de decodificaciones satisfactorias. Es una de las principales causas de penalización del rendimiento de la red.

$$DecodingRate = \frac{\text{Total paquetes decodificados Rx}}{\text{Total paquetes codificados Rx}} \tag{5.3}$$

- *Buffer Size*. Recoge el número total de paquetes que va a poder mantener el *Coding Node*. Un aumento de este parámetro implica un incremento de las oportunidades de codificación.
- *Buffer Time*. Indica el tiempo, en milisegundos, que los paquetes nativos pueden mantenerse en el buffer de codificación. Al aumentarlo, se deberían incrementar las oportunidades de codificación. Por tanto, la obtención de un valor óptimo, al igual que el *Buffer Size*, son dos de las principales dificultades que surgen al configurar NC.
- *Delay*. Parámetro que indica el retardo medio entre paquetes. Para su cálculo se registran todos los instantes temporales de recepción correcta en un array para recoger el valor medio correspondiente en la traza resumen.
- *Jitter*. Caracteriza la variación en el tiempo de llegada de los paquetes, causada por congestión de red, pérdidas de transmisión o por las características de las diferentes rutas seguidas por los paquetes para llegar al destino.
- Número de transmisiones. Variable que indica el número de transmisiones que se han realizado para enviar un único paquete. Es importante para estimar el ahorro energético que supone emplear NC.

## 5.2 Proceso de medida

En esta sección se describen las diferentes decisiones que se han tomado para obtener los parámetros anteriormente mencionados en los tres escenarios analizados.

- Se han realizado un total de 100 ejecuciones independientes sobre cada uno de los escenarios simulados, de tal forma que en las gráficas posteriores se muestra el valor medio de todas estas medidas, acompañadas del intervalo de confianza del 95 %, obtenido mediante la siguiente expresión:

$$Confidence\ Interval = 2{,}2281 \cdot \frac{\sqrt{\text{Varianza de las medidas}}}{\sqrt{\text{Número de medidas}}} \tag{5.4}$$

- Todos los paquetes tendrán el tamaño máximo de datagrama permitido por la conexión, que está fijado en 1500 bytes. Por tanto, teniendo en cuenta que la cabecera de Network Coding tiene como máximo de 35 bytes, ya que como maximo tendrá dos paquetes codificados, si se resta la cabecera IP(20 bytes) y UDP(8 bytes), los datos tienen un tamaño de 1437 bytes.
- Todas las transmisiones se hacen a la velocidad máxima de trabajo de 11 Mbps, mientras que la tasa básica es de 1 Mpbs, emulando la tecnología IEEE 802.11b
- Retransmisiones MAC 802.11. Cuando el canal es hostil se realizan un total de tres retransmisiones antes de descartar el datagrama.

Todas estas características van a ser comunes en todas las medidas que se lleven a cabo. Como ya se ha mencionado, el resto de parámetros se irán modificando durante las diversas simulaciones

## 5.3 Escenarios estudiados

Esta sección describe los escenarios utilizados en cada simulación. Como se ha mencionado se ha trabajado con tres. Se empezará con la topología más sencilla: la cadena. Y se irá incrementando su grado de dificultad, con la *X* y *Butterfly*.

### 5.3.1. Topología lineal: *3 nodos*

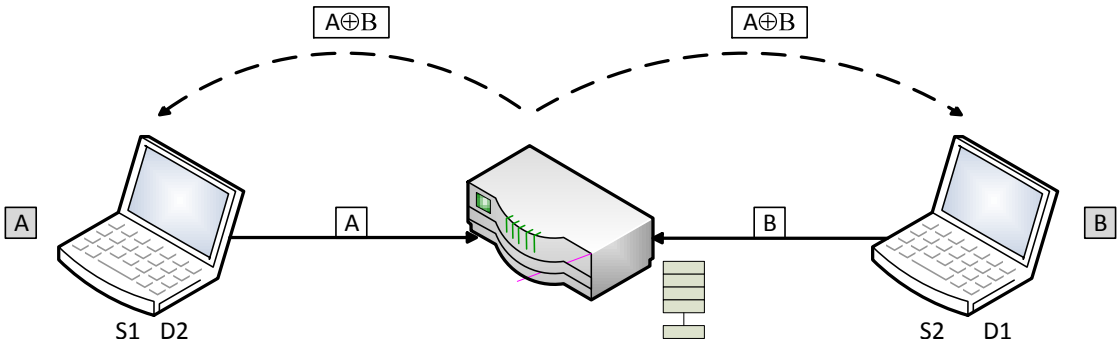



Figura 5.1: Topología lineal

Como se ve en la Figura 5.1, esta topología cuenta con dos equipos que se comunican a través de un único nodo central, que realizará las labores del *Coding Node*. Como se observa se consideran dos flujos, S1-D1 y S2-D2. La distancia a la que se han dispuesto los nodos es la suficiente como para permitir que $S_i$ esté fuera del área de cobertura de $D_i$, pero dentro de la del *Coding Node*. La finalidad es que los $D_i$ sean capaces de realizar el *overhearing* de los paquetes enviados por el nodo central, ya que en este escenario los destinos no tienen que hacer la tarea de *overhearing* de los paquetes nativos, puesto que los emisores ya han guardado en su *Decoding Buffer* el paquete que han enviado previamente.

A continuación se describe paso a paso el funcionamiento de la implementación de NC en este escenario:

1. Los $S_i$ realizan el envío de sus paquetes. El primero que alcance el *Coding Node* se depositará en su buffer, a la espera de un paquete de distinto flujo. Los $D_i$ almacenan el paquete nativo que han transmitido en el *Decoding Buffer*, a la espera de su labor de decodificación posteriormente.

2. Cuando el segundo paquete llega al *Coding Node*, se realiza la codificación de ambos paquetes por medio de una operación binaria XOR, generando un único paquete al

que se le añade su respectiva cabecera NC modificada. Este es enviado en el modo *pseudo-broadcast*

3. Los $D_i$ llevan a cabo el *overhearing*. Cuando un paquete es detectado, lo interceptan, analizan la cabecera NC para saber si son receptores de alguno de los paquetes que forman parte del codificado, comprobando la IP y posteriormente, comprueban que el codificado conste de alguno de los que contiene el *Decoding Buffer*. En este punto se realiza la posterior decodificación mediante la operación XOR. Finalmente, se eliminan las capas prescindibles en el nivel de aplicación ,obteniendo la información nativa que $S_i$ pretendia transmitir a $D_i$.

### 5.3.2. Topología *X*

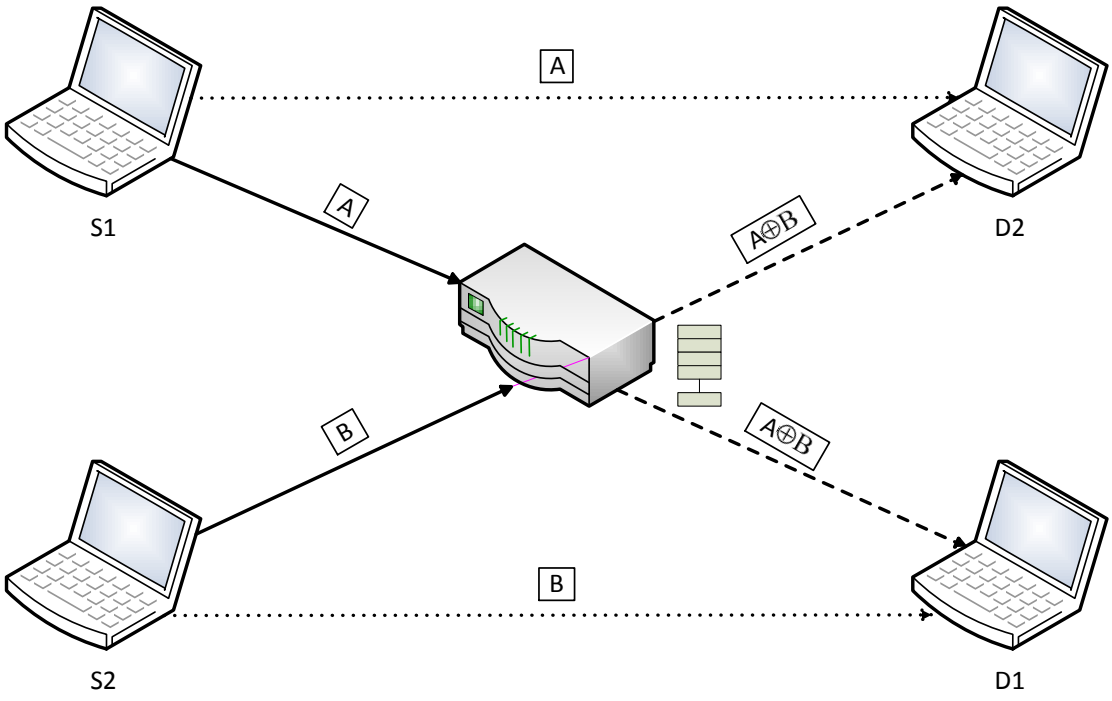



Figura 5.2: Topología X

En este segundo caso la topología cambia, al tener cuatro equipos, $S_1$ y $S_2$ , como emisores, y $D_1$ y $D_2$, como receptores. Se mantiene el número de flujos, pasando ambos por el nodo central, que actuará como *Coding Node*. La distancia a la que se han posicionado los nodos es la suficiente como para permitir que $D_i$ esté fuera del área de cobertura de $S_i$ pero dentro de la correspondiente a $S_j$, con $i, j \in [1, 2]$ e $i \neq j$. Se necesita que los nodos receptores, $D_i$, dispongan de los paquetes nativos enviados desde $S_j$ hacia $D_j$.

A continuación se muestra el funcionamiento de NC en este escenario.

1. Los equipos transmisores, $S_1$ y $S_2$, efectuan el envío de sus respectivos paquetes, A y B, a los receptores, $D_1$ y $D_2$, que almacenan el paquete nativo, A y B, gracias a la técnica de *overhearing*. En los *Decoding Buffer* de $D_1$ y $D_2$ se depositan B y A, respectivamente.

2. Cuando uno de los paquetes (A) alcanza el *Coding Node* se almacena en el *Coding Buffer* y el temporizador comienza a funcionar. Supuesto que el segundo paquete en llegar es B y lo hace antes de que el tiempo expire se estarán dando las condiciones necesarias para llevar a cabo lacodificación. Dicha operación se efectuará con sus respectivas modificaciones en la cabecera NC y, finalmente, será enviado por el canal mediante la técnica *pseudo-broadcast*.

3. Los equipos receptores, $D_1$ y $D_2$, realizan sus operaciones de *overhearing*. Cuando un paquete es detectado, lo interceptan, analizan la cabecera NC para saber si son receptores de alguno de los paquetes que forman parte del codificado (comprobando la IP) y , posteriormente, si el paquete codificado consta de alguno de los que contiene el *Decoding Buffer*, se realiza la decodificación. Una vez eliminadas las cabeceras, el paquete se envía a la aplicación.

### 5.3.3. Topología *Butterfly*

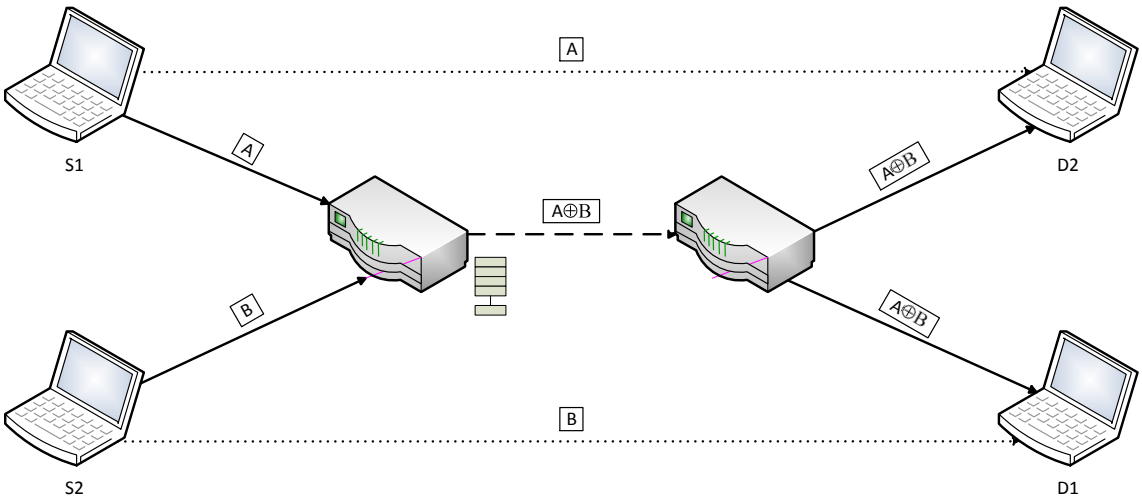



Figura 5.3: Topología Butterlfy

Este último tercer escenario es muy similar al anterior, volviéndose a generar los dos flujos UDP entre los mismos equipos. La diferencia entre estos dos escenarios se situa en un nuevo nodo intermedio cuya función es meramente *store-and-forward*. Por tanto las labores de *overhearing* de los nodos receptores sólo cambiaran respecto al *Coding Node*, que no se encontrará dentro del área de cobertura de éstos, mientras que el nuevo nodo intermedio sí lo estará. Por el resto, el funcionamiento de NC en esta topología es

prácticamente igual al anterior.

La razón por la que no se elige el nuevo nodo como codificador es que por él pasarán todos los paquetes y, si fuese el encargado de la codificación, disminuiría el rendimiento ya que en principio debería ser el primer nodo intermedio de la red el que debe encargarse de la codificación, pues el ahorro en el número de transmisiones es mayor. El tiempo aumentaría al tener que pasar cada paquete de uno en uno por el nodo 1 y despues almacenarse en el nodo 2; se incrementarían los retardos y, por tanto, disminuiría el throughput.

## 5.4 Resultados de las simulaciones

A lo largo del desarrollo de este trabajo se han hecho multitud de medidas que en este capítulo se sintetizan. Por tanto, se van a mostrar los parámetros considerados más relevantes, de forma que pueda apreciarse su influencia sobre la técnica de *Network Coding*. Se ha organizado el estudio de los diversos parámetros por diferentes secciones.

### 5.4.1. Network Coding en un canal ideal(FER=0.0)

En primer lugar se van a ilustrar las beneficios de NC en condiciones ideales. Como se ha visto en el Capítulo 4, se varia la tripleta de parámetros [FER, *BufferSize*, *BufferTime*]. Así que se ha fijado la FER a cero, se analiza la influencia de la modificación de los otros dos sobre el rendimiento final del tráfico UDP.

A continuación se muestra los resultados, parámetro a parámetro, para cada uno de los tres escenarios.

#### *Coding Rate y Decoding Rate*

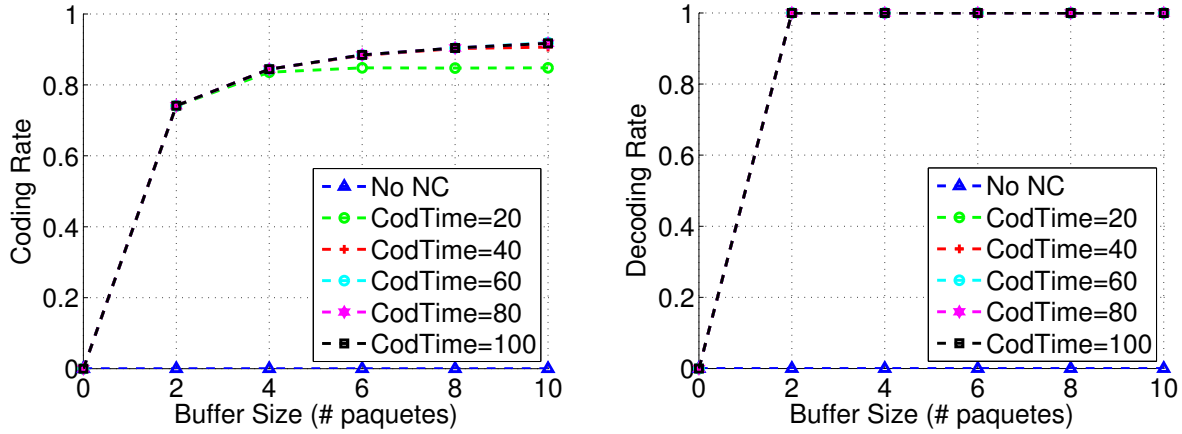
Como se puede observar en las Figuras 5.4a y 5.4b, en un entorno ideal (FER=0.0), el aumento del tamaño del buffer del *Coding Node* produce un incremento notable del *Coding Rate* en los tres escenarios, ya que, al tratarse UDP de un protocolo no orientado a conexión, el aumentar *Buffer Size* y *BufferTime* tiene como consecuencia la aparición de un gran número de oportunidades de codificación.

Se podría pensar por otro lado que un valor grande del *BufferTime* tendría como consecuencia un aumento del retardo (una disminución del throughput), pero el hecho de que no haya errores (FER=0.0), el incremento de este parámetro no penaliza el *Coding rate*, ni el *Decoding Rate*, ni el *throughput*, como se verá posteriormente, ya que los dos flujos se transmiten correctamente en todo momento.

Por otra parte, los resultados del *Decoding Rate* son también esperados, porque las simulaciones siempre se han realizado con un número máximo de paquetes codificados de dos, por lo que, en cuanto el tamaño del BufferSize alcanza tal valor, *Decoding Rate* vale

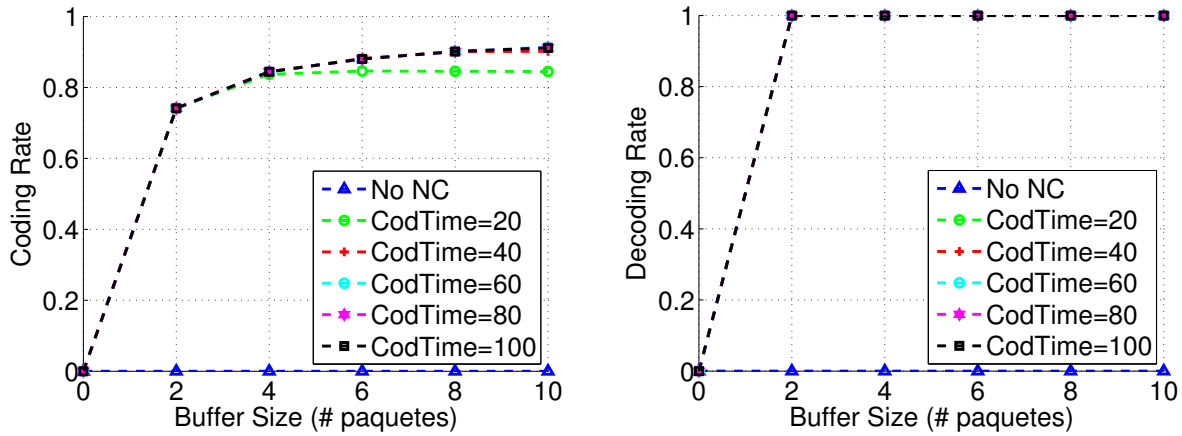
1, pues la decodificación es siempre exitosa.

Nótese que con *BufferSize* o *BufferTime* nulos la transmisión es la trandicional, en la que no se realiza ningún tipo de codificación.



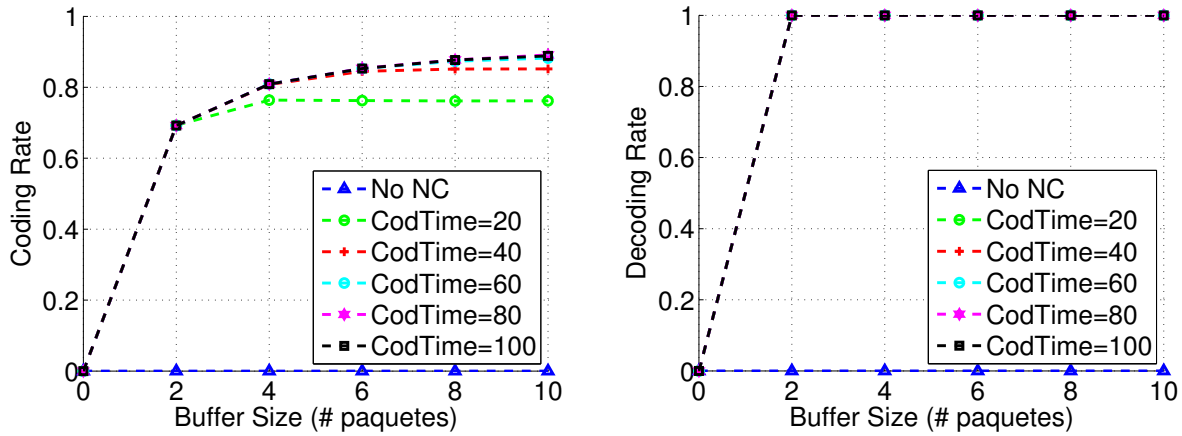

(a) Coding Rate (lineal) (b) Decoding rate (lineal)

Figura 5.4: FER=0 variando BufferSize y BufferTime con topología *lineal*




(a) Coding Rate (X) (b) Decoding rate (X)

Figura 5.5: FER=0 variando BufferSize y BufferTime con topología *X*




(a) Coding Rate (Butterfly)

(b) Decoding rate (Butterfly)

Figura 5.6: FER=0 variando BufferSize y BufferTime con topología *Butterfly*

***Throughput***

A continuación se analiza uno de los parámetros más relevantes, como es el rendimiento de la red. Su incremento es una de las principales bondades de la implementación de NC, uno de los objetivos que se esperaba encontrar a lo largo de este trabajo.

Se va a utilizar la siguiente expresión para obtener datos númericos de la mejora que realiza NC en los tres escenarios:

$$\Delta NC = \frac{\text{Thput con NC} - \text{Thput sin NC}}{\text{Thput sin NC}} \times 100 \tag{5.5}$$

$$\Delta NC_{lineal} = \frac{2{,}2114 - 1{,}5657}{1{,}5657} \times 100 = 41{,}42\,\% \tag{5.6}$$

$$\Delta NC_{X} = \frac{2{,}2897 - 1{,}1815}{1{,}6815} \times 100 = 44{,}78\,\% \tag{5.7}$$

$$\Delta NC_{Butterfly} = \frac{1{,}7332 - 1{,}0780}{1{,}0780} \times 100 = 60{,}77\,\% \tag{5.8}$$

La configuración empleada para estos cálculos ha la que se corresponde con un BufferSize y BufferTime óptimos para las tres topologías. Un paquete codificado reduce siempre una transmisión por cada nodo por el que pasa, lo que se traduce en un mayor tiempo para que los nodos puedan enviar otros paquetes. Así, en la topología *lineal* y en la $X$ se ahorra una transmisión por el envío simultáneo (sólo existe un nodo intermedio), mientras que en Butterfly la diferencia es que se logran evitar dos transmisiones, al haber dos nodos intermedios.

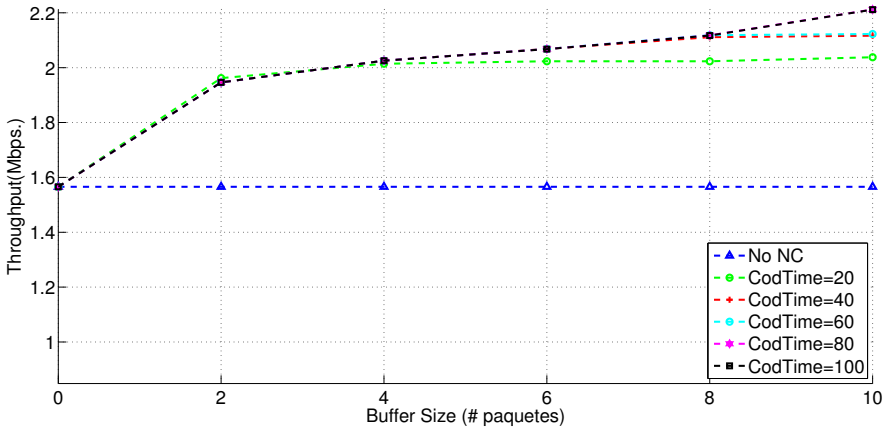



Figura 5.7: Throughput(lineal)

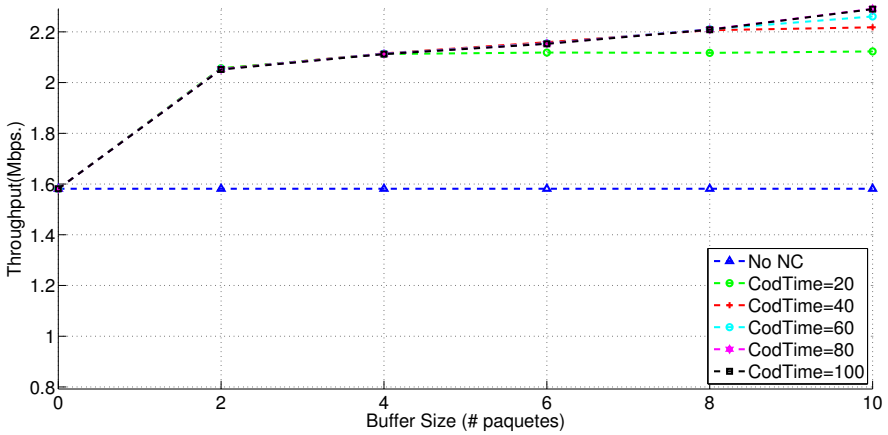



Figura 5.8: Throughput(X)

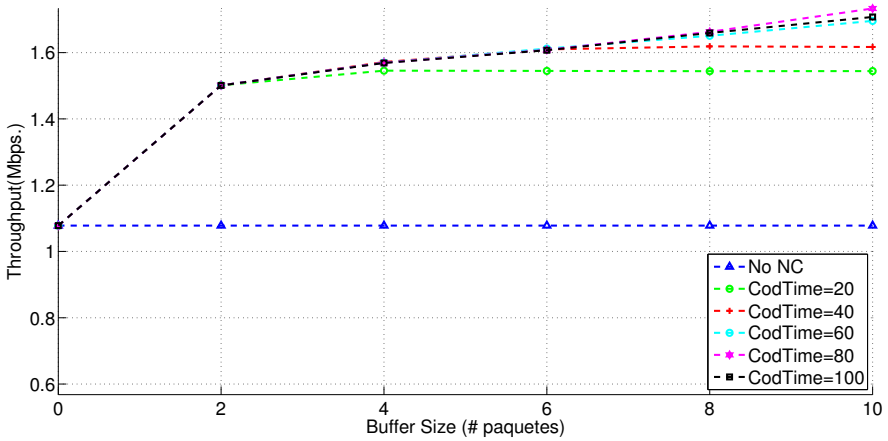



Figura 5.9: Throughput(Butterfly)

En la configuración lineal, con el valor máximo de *BufferSize* y, para *BufferTime* mayores de 80 ms. se obtienen los resultados óptimos como se ve en la Figura 5.7. Lo mismo se da con la topología de la *X* (véase Fig. 5.8), mientras que en el escenario *butterfly* el mejor valor se corresponde con 80 ms. Como se explicó anteriormente, este último cuenta con un nodo intermedio más, lo que aumenta el retardo, y disminuye el *throughput*, como se ve en la Figura 5.9.

Al trabajar sobre el protocolo UDP en los tres escenarios, los mejores resultados de rendimiento se dan cuando los parámetros de configuración toman sus valores máximos.

### *Número de transmisiones*

A lo largo del documento, se han mencionado los beneficios de *Network Coding*, entre los que destaca el ahorro energético. Se ha visto que el número de transmisiones disminuía al emplear NC en un escenario lineal y en la topología X. Ahora se analiza este aspecto, comprobándose que disminuye de forma drástica en los tres escenarios (Figuras 5.10, 5.11 y 5.12).

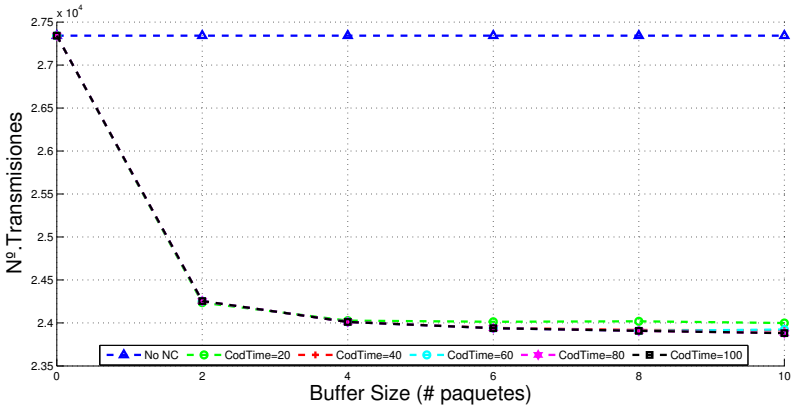



Figura 5.10: Nº.Transmisiones (lineal)

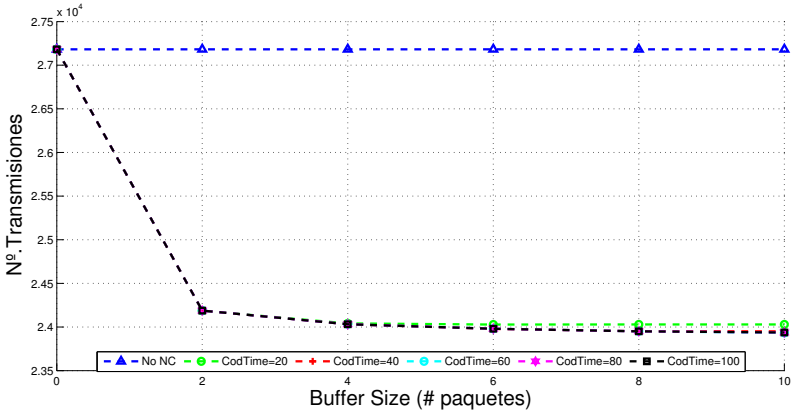



Figura 5.11: Nº.Transmisiones (x)

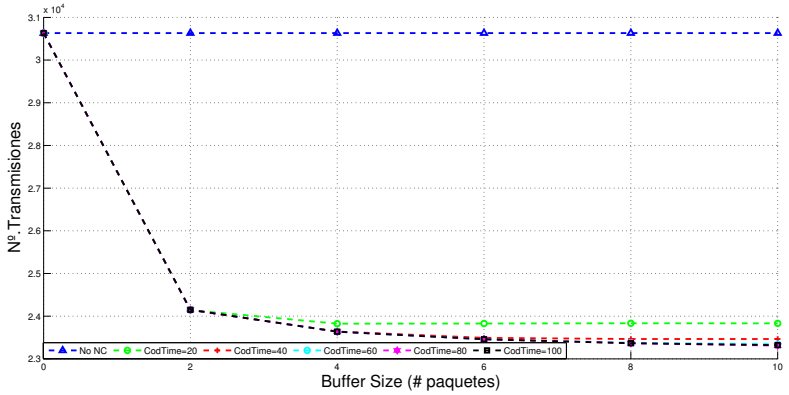



Figura 5.12: Nº.Transmisiones (butterfly)

### 5.4.2. Estudio de los diferentes tipos de canal

Como se ha mencionado se han dispuesto de tres configuraciones de canal (Normal, Middle y Side), sobre los que se han realizado las siguientes medidas, variando la FER: Coding Rate, Decoding Rate, Throughput, Retardo y Jitter. Se analizará cada configuración individualmente, para finalmente comparar los resultados de los tres escenarios. Para la exposición de los resultados se ha elegido manener el BufferSize a diez paquetes y el BufferTime a 100 milisegundos, ya que ofrecieron un comportamiento óptimo sobre el canal ideal.

#### Configuración Normal

Esta configuración de canal es la más realista, pues todos los enlaces de los que consta el escenario son propensos a la existencia de errores (FER), por lo que un paquete codificado puede perderse durante su transmisión por los enlaces intermedios, o bien a la existencia de fallo en el proceso de *overhearing* que impida la decodificación en uno de los destinos.

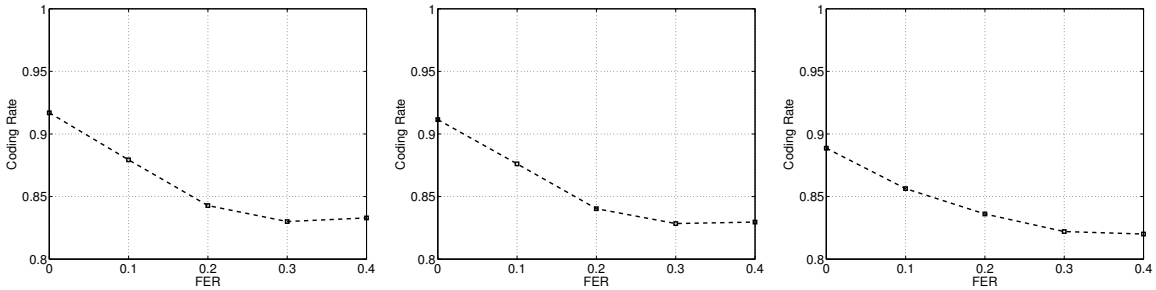
1. *Coding Rate*: se observan las limitaciones que presenta el canal a medida que la FER aumenta. En la Figura 5.13a se ve que en los tres escenarios, este parámetro sufre un descenso, como consecuencia del incremento del número de errores en el canal inalámbrico. Cuando uno de los transmisores envía más información que el otro el número de oportunidades de codificación disminuye.

2. *Decoding Rate*: como ya se ha mencionado, en el escenario lineal no hace falta realizar un proceso de *overhearing* para obtener los paquetes nativos, pues en este escenario tan sencillo ya están inicialmente en el *Decoding Buffer*. Por tanto, nunca se dará la situación de no disponer de los paquetes nativos para la posterior decodificación. En los otros dos escenarios, la pérdida de paquetes afecta muchas veces a un sólo flujo, a excepción de que sea en el enlace intermedio del escenario Butterfly, o que ninguno de los destinos reciba el paquete codificado. En estos casos también se suma la posibilidad de la no existencia de los nativos en el *Decoding Buffer*, lo que disminuye de manera significativa el *Decoding Rate*.

3. *Delay*. Las Figuras 5.15 ponen de manifiesto como el tiempo de llegada entre paquetes aumenta de un escenario a otro, ya que se pasa de un escenario muy sencillo a un segundo, la topología X, en el que aparece un nodo intermedio y, por ultimo, al escenario Butterfly, en el que el retardo vuelve a aumentar a consecuencia del enlace intermedio entre los nodos. Además, se ve como la FER que penaliza las transmisiones, de forma que al incrementar la pérdida de paquetes, aumenta el retardo, por las retransmisiones a nivel MAC 802.11

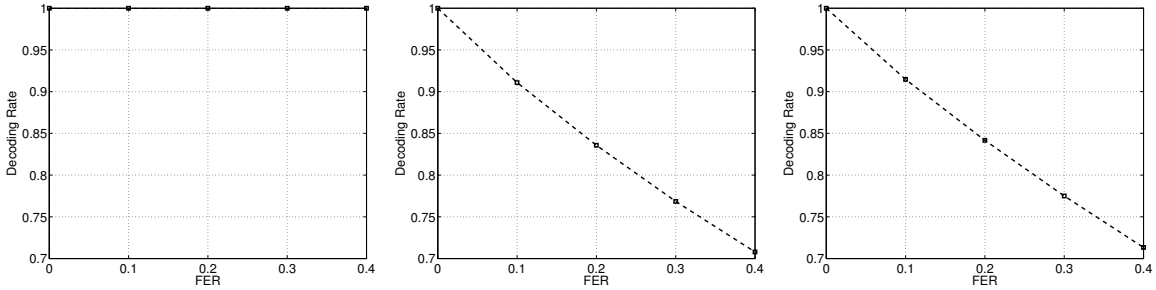
4. *Jitter*: respecto a este parámetro que, a grandes rasgos, da idea de la variación del retardo, se ve un aumento a medida que el número de errores en el canal es mas apreciable.

5. *Throughput*: al dejar el parámetro con más relevancia para el final, se puede pueda observar la influencia del resto de variables sobre él. Por una parte, el aumento de fallos en la decodificación se traduce en una pérdida de datos útiles y, por otra, la disminución de oportunidades de codificación, junto a la pérdida de paquetes que derivan en las retransmisiones a nivel MAC 802.11 implican un incremento del retardo. Todo ello va a disminuir notablemente el cociente del rendimiento, como se ve en las Figuras 5.17. Nótese que en el escenario lineal, el *Decoding Rate* siempre es elevado, alcanzando su valor máximo, con lo que se consigue un rendimiento mayor que en las otras topologías al aumentar la FER.



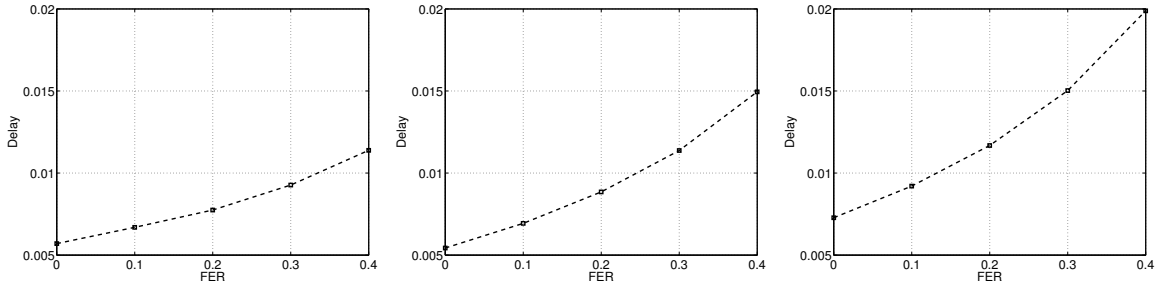

(a) Coding Rate (lineal) (b) Coding Rate (X) (c) Coding Rate (Butterfly)

Figura 5.13: Coding Rate Vs. FER con BufferSize = 10 y BufferTime = 100 ms.



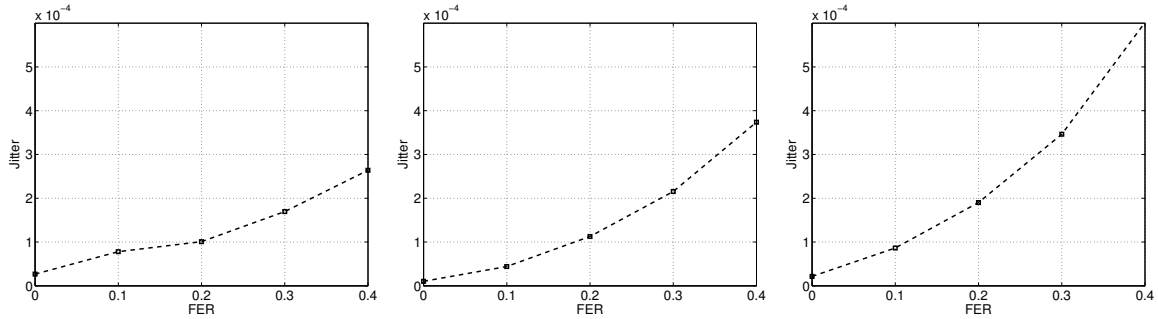

(a) Decoding Rate (lineal) (b) Decoding Rate (X) (c) Decoding Rate (Butterfly)

Figura 5.14: Decoding Rate Vs. FER con BufferSize = 10 y BufferTime = 100 ms.



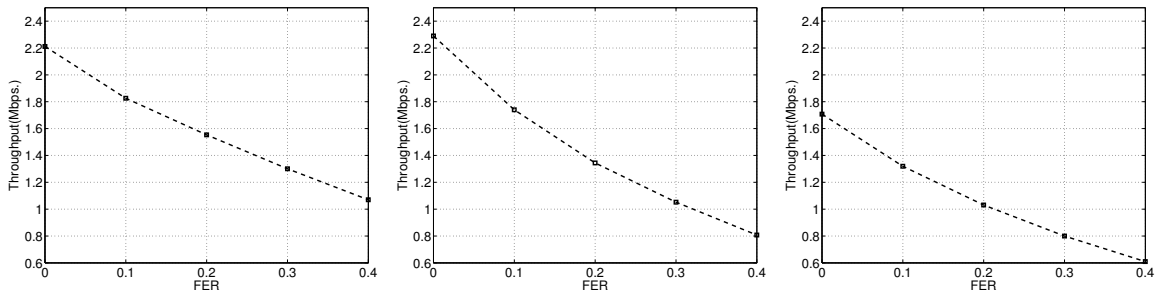

(a) Delay (lineal) (b) Delay (X) (c) Delay (Butterfly)

Figura 5.15: Delay Vs. FER con BufferSize = 10 y BufferTime = 100 ms.




(a) Jitter (lineal) (b) Jitter (X) (c) Jitter (Butterfly)

Figura 5.16: Jitter Vs. FER con BufferSize = 10 y BufferTime =100 ms.




(a) Throughput (lineal) (b) Throughput (X) (c) Throughput (Butterfly)

Figura 5.17: Rendimiento Vs. FER con BufferSize = 10 y BufferTime = 100 ms.

### Configuración Side

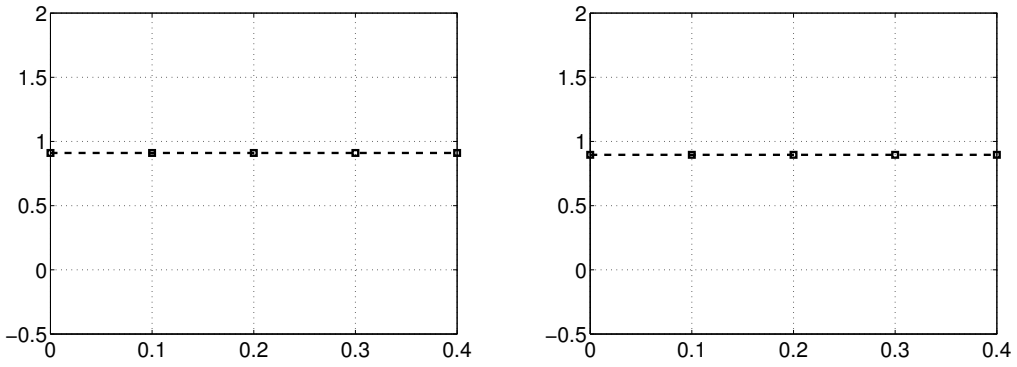
En esta segunda configuración se estudia la influencia de la pérdida de paquetes en los enlaces extremos sobre el rendimiento de NC , que se utilizan para almacenar los paquetes nativos en el *Decoding Buffer* del receptor mediante la técnica de *overhearing* por parte de los nodos receptores. A partir de ahora no se analiza escenario lineal, ya que no se puede utilizar dicha configuración.

1. *Coding Rate*: el *Coding Node* recibe los paquetes nativos a través de un canal libre de errores, por lo que el parámetro que se está estudiando se va a mantener constante a un valor próximo a la unidad, lo que se traduce en que la mayor parte de los paquetes transmitidos son codificados (incremento de las oportunidades de codificación) como se puede ver en la Figura 5.18.

2. *Decoding Rate*: aunque los paquetes que atraviesan el nodo intermedio van alcanzar de manera satisfactoria su destino, al no producirse errores en dicho canal, lo que no significa que la decodificación vaya a ser adecuada, ya que al encontrar errores en los canales sobre los que se hacer *overhearing* no siempre se va a contar con los paquetes nativos en el *Decoding Buffer*, lo que va a suponer una disminución progresiva de este parámetro al incrementar FER, como se observa en la Figura 5.19. Esto penaliza notablemente el rendimiento, puesto que la pérdida de de oportunidades de codificación se traduce en pérdida de datos.

3. *Delay* y *Jitter*: al igual que en la configuración normal, estos parámetros suben a medida que la FER crece. Es verdad que en esta configuración se obtienen mejores resultados que con la anterior, debido a que no existen errores en el canal intermedio.

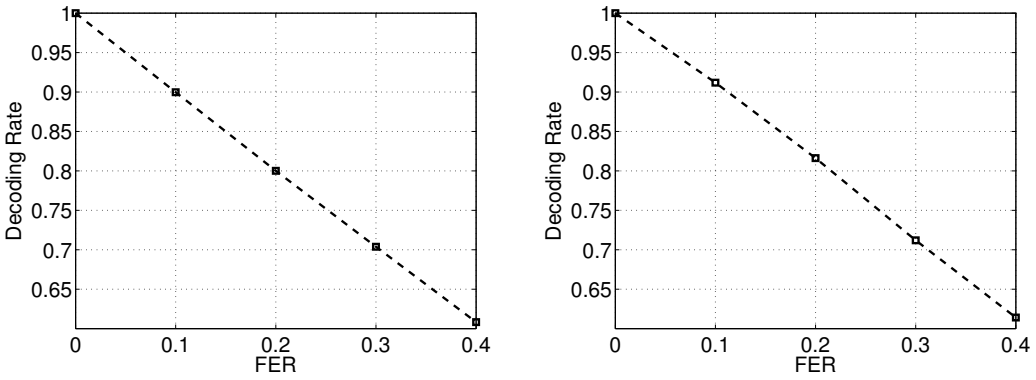
4. *Throughput*: se aprecia que la tendencia de las gráficas de la Figura 5.22 es similar a las del *Decoding Rate* (Figura 5.19), lo que se debe a que este último parámetro penaliza de manera notable el rendimiento en NC. Como la FER sólo genera errores en los enlaces extremos, tiene mayor influencia sobre el *throughput* la pérdida de datos de las decodificaciones no satisfactorias que el retardo medio.




(a) Coding Rate (X)

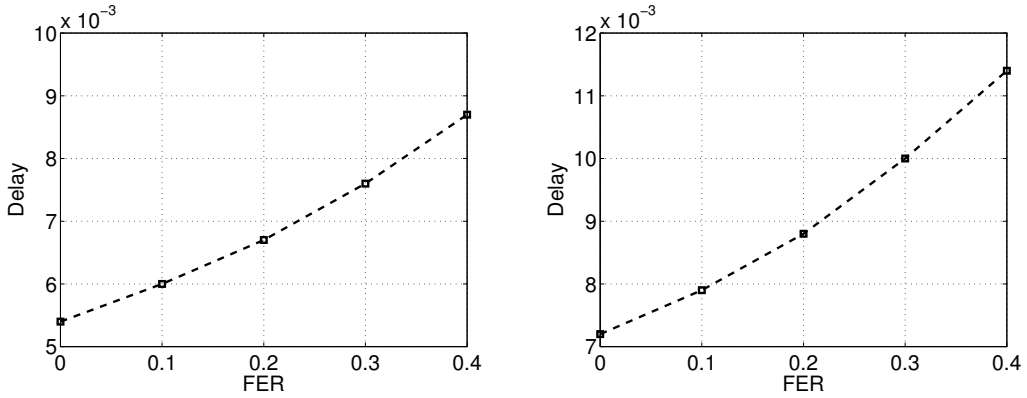
(b) Coding Rate (Butterfly)

Figura 5.18: Coding Rate Vs. FER con BufferSize = 10 y BufferTime = 100 ms.



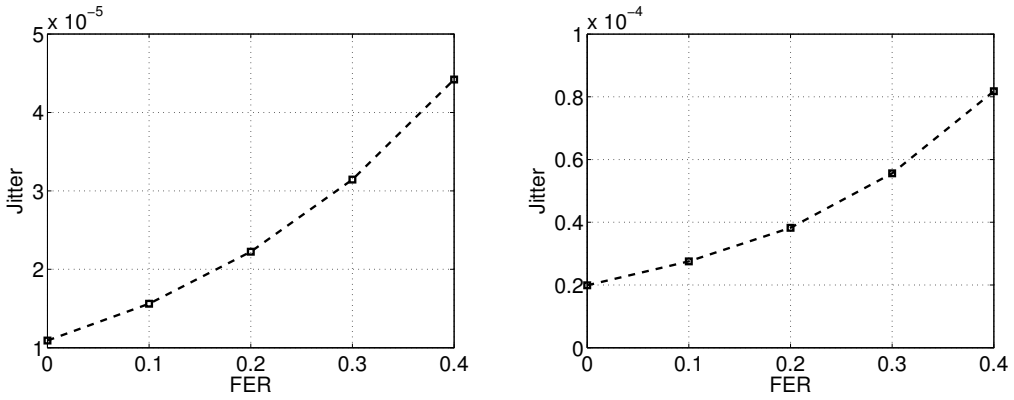

(a) Decoding Rate (X) (b) Decoding Rate (Butterfly)

Figura 5.19: Decoding Rate Vs. FER con BufferSize=10 y BufferTime=100ms.



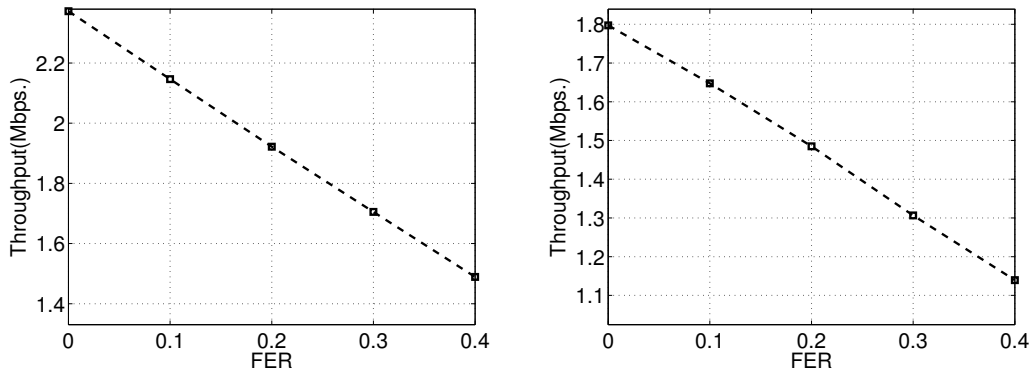

(a) Delay (X) (b) Delay (Butterfly)

Figura 5.20: Delay Vs. FER con BufferSize=10 y BufferTime=100ms.




(a) Jitter (X) (b) Jitter (Butterfly)

Figura 5.21: Jitter Vs. FER con BufferSize=10 y BufferTime=100ms.




(a) Throughput (X) (b) Throughput (Butterfly)

Figura 5.22: Rendimiento Vs. FER con BufferSize=10 y BufferTime=100ms.

### Configuración Middle

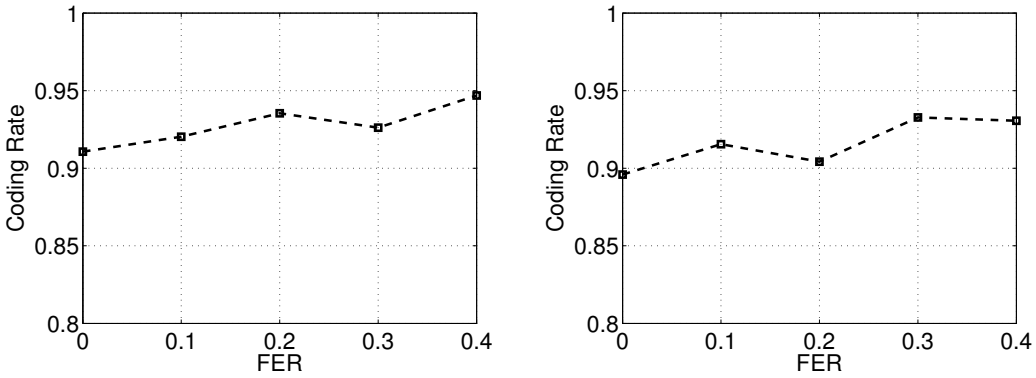
En esta tercera configuración los errores se van a posicionar en el enlace intermedio, por lo que producirán pérdidas bien desde el *Coding Node* hacia sus destinos en la topología X o escenario butterfly en el enlace entre los nodos intermedios. Aquí no se va a dar el problema anterior, porque siempre se dispone de los paquetes nativos necesarios para realizar la decodificación de los paquetes. El conflicto yace en la pérdida de paquetes codificados que, además, es diferente en ambos escenarios, ya que en X no hay un tramo común por lo que si se pierde un paquete codificado puede afectar a uno de los dos flujos. En la topología butterfly si se pierde un paquete codificado afecta a ambos flujos, por el enlace común ya mencionado.

1. *Coding Rate*: el *Coding Node* recibe los paquetes nativos a través de un canal libre de errores. Por tanto se genera un alto número de oportunidades de codificación como se refleja en la Figura 5.23.

2. *Decoding Rate*: mide las oportunidades de decodificación satisfactorias que se hayan dado en el receptor. Los nodos destinos siempre van a disponer de los paquetes nativos ya que el proceso de *overhearing* es siempre satisfactorio, por el hecho de la no existencia de errores en el canal, así, siempre que un paquete codificado alcance uno de estos receptores, la decodificación se producirá sin problemas. Véase la Figura 5.24.

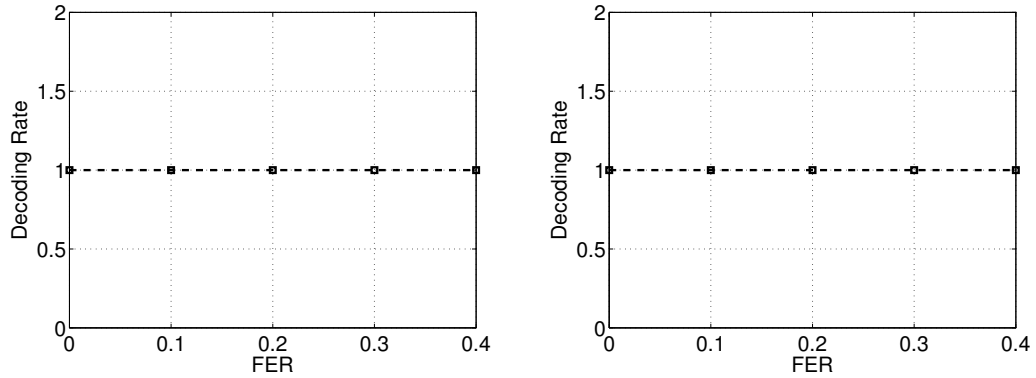
3. *Delay* y *Jitter*: la pérdida de paquetes codificados en los enlaces intermedios se va a traducir en un aumento de estos parámetros, debido a las retransmisiones a nivel MAC 802.11. Además, produce una pérdida de datos útiles, lo que también disminuirá el rendimiento.

4. *Throughput*: se ve en la Figura 5.27 que el incremento del retardo producido por la disposición de errores en los canales intermedios tiene como consecuencia una importante disminución del rendimiento de la red.



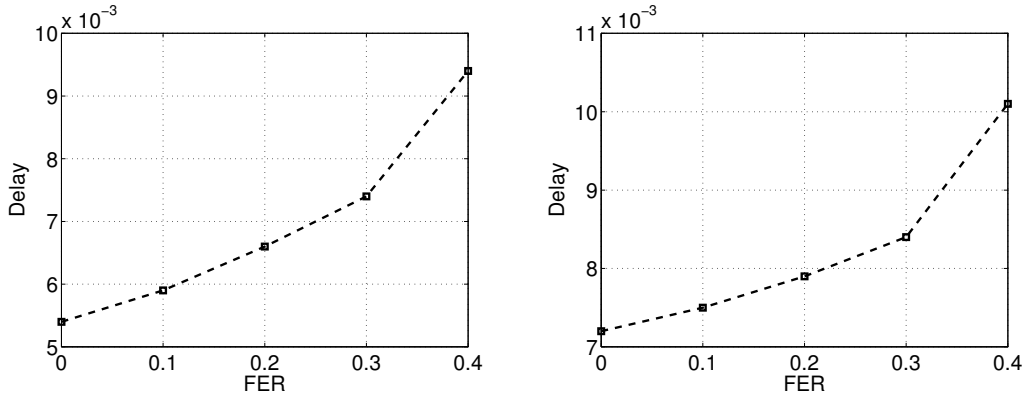

(a) Coding Rate (X) (b) Coding Rate (Butterfly)

Figura 5.23: Coding Rate Vs. FER con BufferSize=10 y BufferTime=100ms.



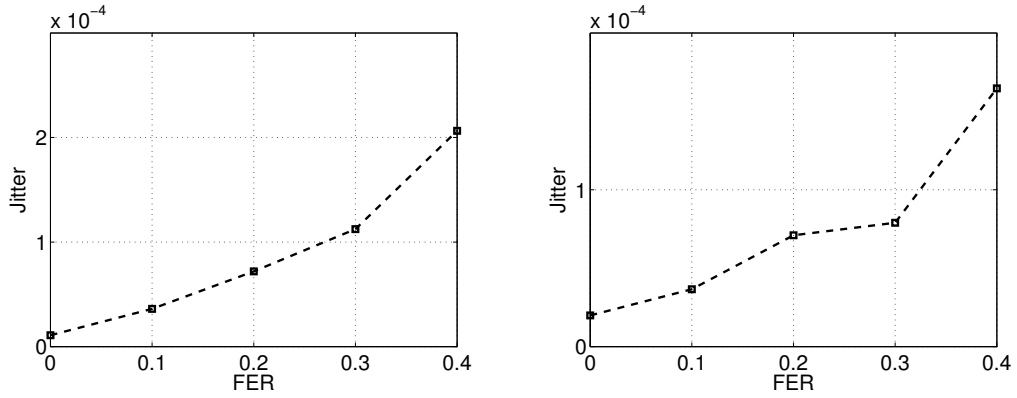

(a) Decoding Rate (X) (b) Decoding Rate (Butterfly)

Figura 5.24: Decoding Rate Vs. FER con BufferSize = 10 y BufferTime = 100 ms.



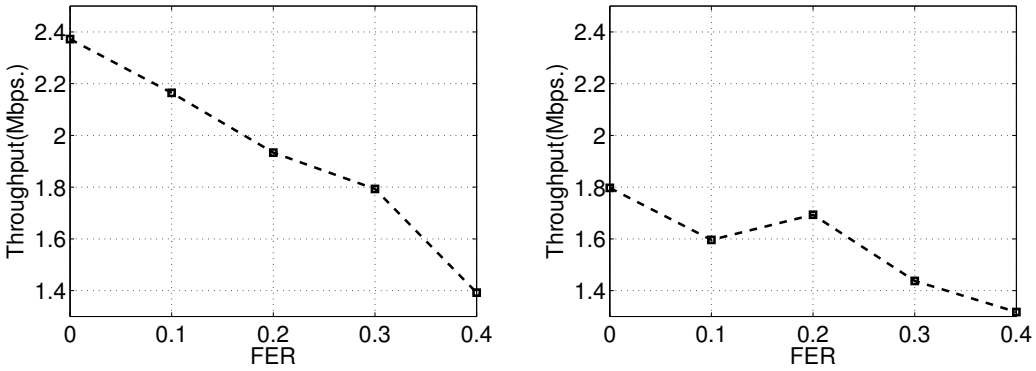

(a) Delay (X) (b) Delay (Butterfly)

Figura 5.25: Delay Vs. FER con BufferSize=10 y BufferTime=100ms.




(a) Jitter (X) (b) Jitter (Butterfly)

Figura 5.26: Jitter Vs. FER con BufferSize=10 y BufferTime=100ms.




(a) Throughput (X) (b) Throughput (Butterfly)

Figura 5.27: Rendimiento Vs. FER con BufferSize=10 y BufferTime=100ms.

### Comparación Global

Esta sección proporciona una visión sintentizada de los resultados obtenidos a lo largo de este proyecto.

- Escenario lineal: se trata de la topología más sencilla, en la que se pueden visualizar de manera directa los beneficios de NC.

  - En lo que se refiere al retardo (Véase Figura 5.28c) el uso de NC da lugar a un descenso de este parámetro, que en todo momento es el mismo respecto a una configuración sin NC como se ve en la Figura 5.28d donde el Jitter, la variación del retardo, con y sin NC, es el mismo.
  - En la Figura 5.28e se muestra un incremento importante del rendimiento, uno de los objetivos principales de este trabajo. El *throughput* sufre dos grandes penalizaciones: *Decoding Rate* y los errores en el canal. Al maximizar las oportunidades de decodificación, manteniendo la misma FER en el canal, se ve la gran ventaja de NC, ya que se está enviando la misma cantidad de datos en menos tiempo, debido al ahorro energético, consecuencia de un menor número de transmisiones, como se vio en la Sección 5.4.1. *Decoding rate* siempre toma su valor máximo ya que en el *Decoding Buffer* siempre están siempre los paquetes nativos para una decodificación satisfactoria, al no tener que realizar el proceso de *overhearing* para contar con ellos.

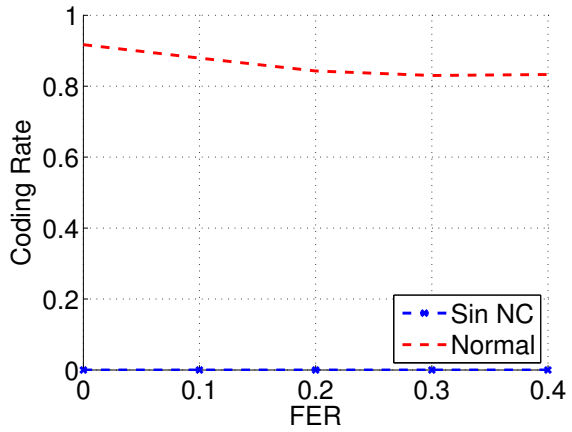
- Escenario X

  - Se ha comprobado, Figura 5.29b, que con una Configuración Middle se obtiene un *Decoding Rate* mayor que en las otros supuestos; esto ocurre porque no va a haber errores en los enlaces extremos por los que los nodos destino realizan el proceso de *Overhearing* y, por tanto, van a tener siempre y en todo momento los paquetes nativos no útiles para la posterior decodificación del paquete codificado, que siempre va a ser satisfactoria. Obsérvese como en este mismo caso, en el que se han dado los mayores niveles de *Coding Rate* y *Decoding rate*, se ha obtenido el *throughput* mayor, porque las decodificaciones van a ser siempre satisfactorias, por lo que el retardo debido a las retransmisiones MAC 802.11 será el único factor que degrade sobre el rendimiento de la red.
  - En cuanto al retardo, se ve que, en una configuración Normal, al encontrar errores en todos los enlaces, se obtienen peores tiempos que en el resto de configuraciones, incluso llegando a superar a una situación sin NC.
  - Finalmente, debido a los errores en el canal, se pone de manifiesto que en una configuración Side o Middle se dan mayores rendimientos, mientras que en una situación más realista, al incrementar la FER, el comportaiento se iguala al de una transmisión tradicional.

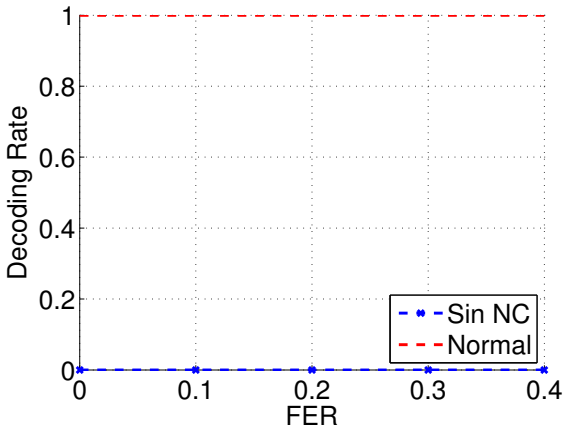
- Escenario Butterfly

  - En esta topología, en cuanto a *Coding Rate* y *Decoding Rate* los resultados son prácticamente iguales que en la topología anterior. Lógicamente, al pasar por un nodo intermedio más, el Retardo y Jitter son mayores, pero se ve que al emplear NC se obtienen mejores resultados.

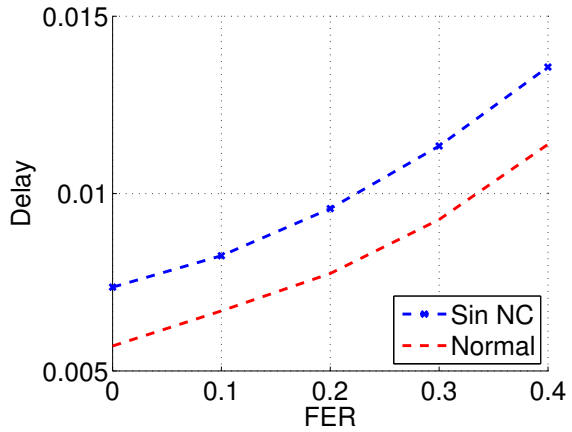
  - El beneficio aparece de manera más clara en las configuraciones,Side y Middle, ya que se obtiene un mayor rendimiento de la red, que siempre va a mantenerse por del de encima que en una configuración Normal y, por supuesto, que el observado en una transmision tradicional sin *Network Coding*. Como se ha mencionado en la anterior topología, en la configuración no realista se consiguen los mayores valores de rendimiento de la red, ya que los paquetes nativos siempre están disponibles en el *Decoding Buffer*, de forma que nunca se pierde una oportunidad de decodificación, una de las grandes causas de penalización en el *throughput*.
  - Por otra parte, la FER la otra penaliza el *throughput* dependiendo de la configuración concreta que se esté analizando. En la Middle, el retardo aparece como el factor que más limita, mientras que en Side, será el *Decoding Rate*. La Figura 5.30e recoge mejores resultados para la Configuración Middle, en la que el rendimiento disminuye debido al retardo y no a la pérdida de datos útiles. Sin embargo, la Configuracion Side es el caso opuesto, ya que hay pérdida de paquetes de datos debido a una disminución en el número de oportunidades de decodificación.



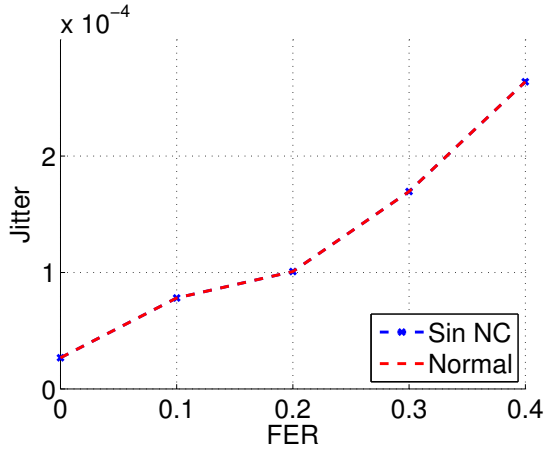

(a) Coding Rate (lineal)



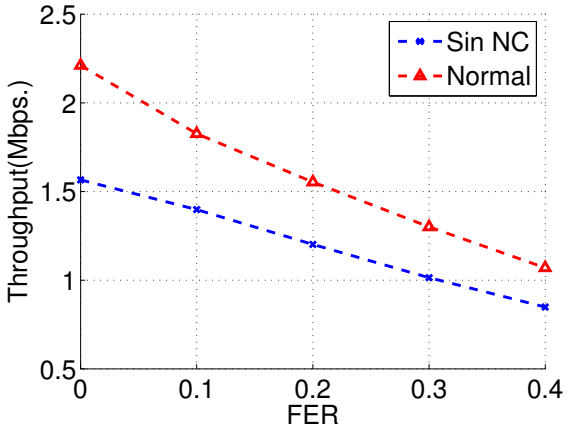

(b) Decoding Rate (lineal)




(c) Delay (lineal)

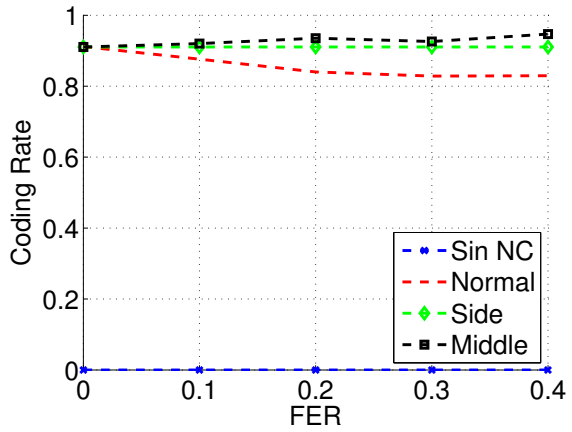



(d) Jitter (lineal)

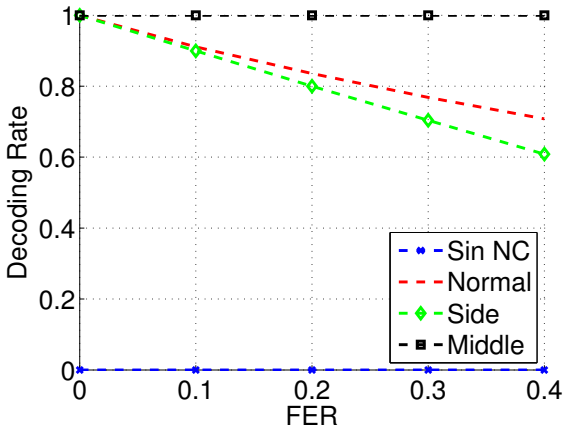



(e) Throughput (lineal)

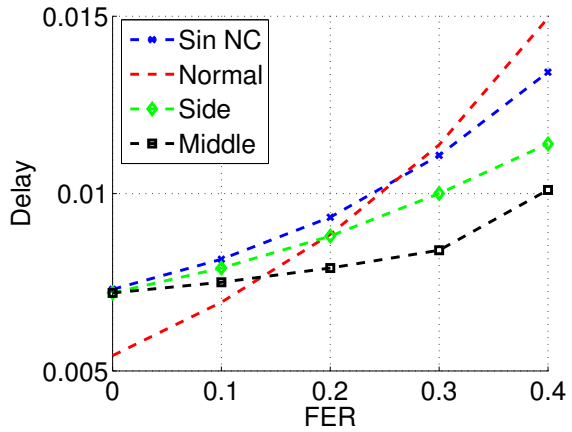
Figura 5.28: Comparación global - Escenario lineal



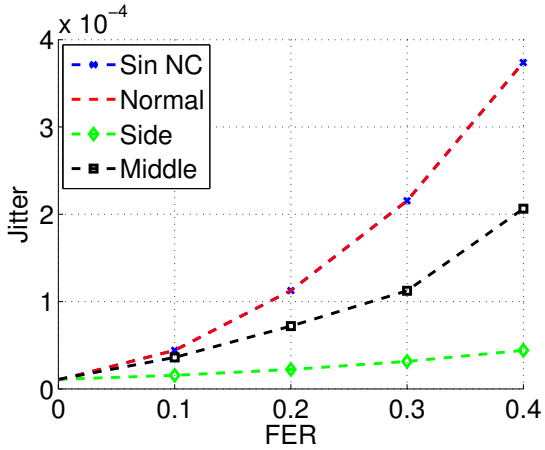

(a) Coding Rate (X)



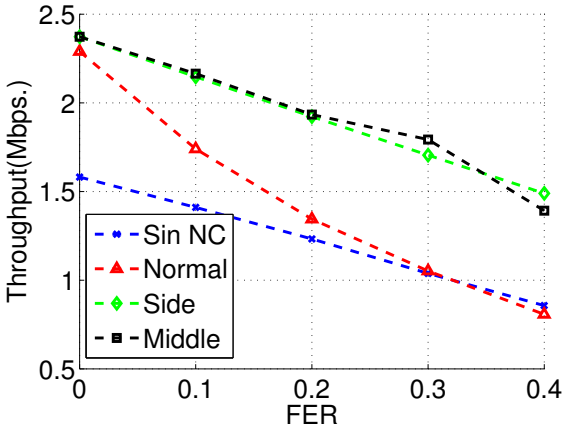

(b) Decoding Rate (X)




(c) Delay (X)

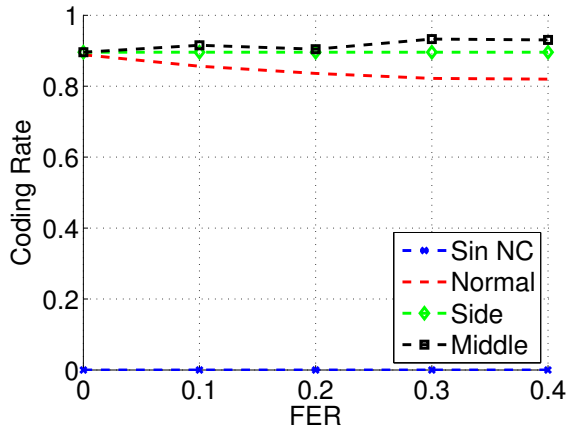



(d) Jitter (X)

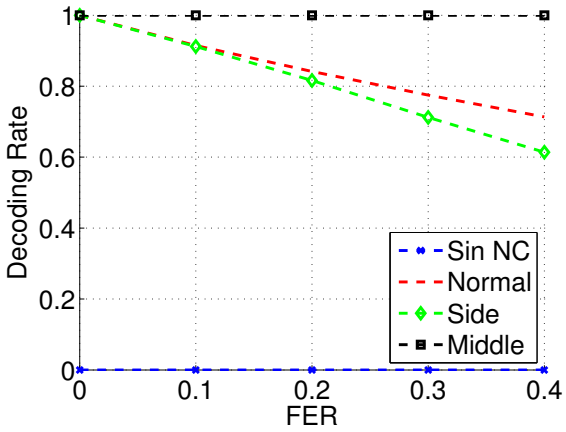



(e) Throughput (X)

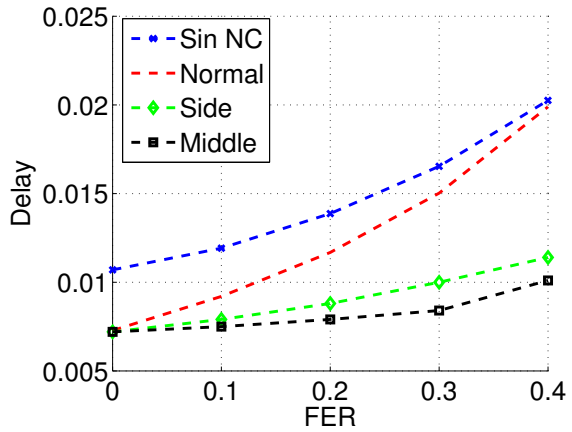
Figura 5.29: Comparación global - Escenario X



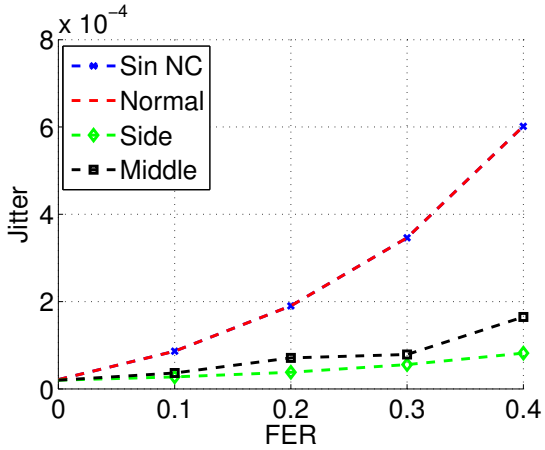

(a) Coding Rate (Butterfly)



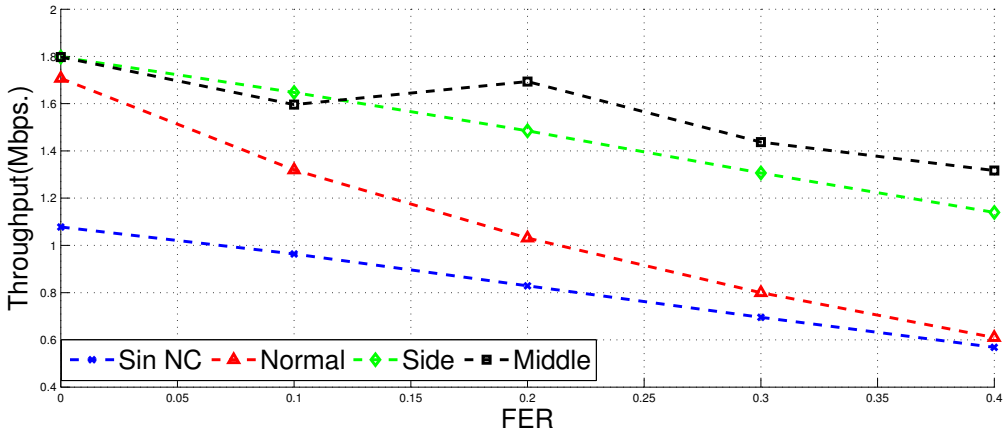

(b) Decoding Rate (Butterfly)




(c) Delay (Butterfly)




(d) Jitter(Butterfly)




(e) Throughput (Butterfly)

Figura 5.30: Comparación global - Escenario Butterfly

# 6 Conclusiones y Líneas futuras

## 6.1 Conclusiones

A continuación se van a destacar varias conclusiones que se pueden extraer a partir de las investigaciones realizadas a lo largo de este trabajo.

Al trabajar en un medio inalámbrico ideal (sin errores) se han puesto de manifiesto de manera clara los beneficios de *Network Coding*. Se han dado incrementos del rendimiento alrededor del 50 % en los mejores casos cuando se ha trabajado con los máximos tamaños de *BufferSize* y *BufferTimer*. Cuando se ha incrementado el grado de error en los enlaces del canal a lo largo de todas las simulaciones se ha visto que, al no existir un control de congestión, casi siempre se obtiene un mejor rendimiento empleando NC. En algunos casos, la topología X (Figura 3.4), se ha observado que a partir de un valor de FER ($\simeq 0.3$) no existe un incremento notable, llegando a encontrar un mayor *throughput* sin NC cuando FER alcanza un valor de 0.4.

El rendimiento de la red disminuye con el incremento de la FER, bien debido a la pérdida de los paquetes nativos que tienen que encontrarse en el *Decoding Buffer*, o a la pérdida de paquetes nativos hacia el *Coding Node*. Todo esto penaliza de forma negativa el rendimiento de la red que, con valores de FER elevados, se acerca a los valores observados en una transmisión tradicional.

*BufferSize* y *BufferTime* son dos parámetros con una influencia notable en el comportamiento global de NC. Con valores altos de ambos parámetros se ha conseguido un mayor número de oportunidades de codificación. Así, se reduce el número de transmisiones y, por tanto, el retardo de todos los datos transmitidos.

Tras todos los estudios realizados, se podría concluir en que el impacto de aplicar técnicas de NC sobre tráfico UDP resulta beneficioso en términos de rendimiento. Se ha visto la mejora que se produce en redes con topologías como la X o *Butterfly*. Es cierto que cuando la FER es elevada, el rendimiento es muy próximo al que existe en en una comunicación tradicional, pero se podría concluir que *Network Coding* aporta una clara mejora para las comunicaciones que utilizan el protocolo UDP.

## 6.2 Líneas futuras

Los resultados obtenidos a lo largo de todo este trabajo son un primer paso del estudio de NC junto a un protocolo de transporte no orientado a conexión. Así, este proyecto deja varias líneas de investigación abiertas.

Se han realizado simulaciones con escenarios sencillos como son el lineal, X y Butterfly. Sería de utilidad llevar a cabo las simulaciones que hemos estudiado sobre topologías más complejas, por ejemplo sobre despliegues aleatorios. En ellas sería necesario tomar una decisión para determinar el nodo que tomará el papel *Coding Node*. Por otra aprte, se tendría que resolver el problema de hacer llegar los paquetes nativos necesarios al *Decoding Buffer*.

Existe un aspecto a nivel de enlace que queda abierto para resolver en futuras implementaciones de NC: en el campo destino de la cabecera MAC 802.11 sólo se indica la dirección MAC de uno de los destinos, como ya se indica al explicar el proceso de *pseudo-broadcast*. Por tanto, sólo el equipo designado en tal cabecera realizará las tareas de confirmación sobre la recepción del paquete. El transmisor no puede saber si el paquete lo han recibido el resto de nodos.

Finalmente, también se debería usar un modelo de canal más realista (errores a ráfagas) en el que se pueda analizar hasta qué punto el uso combinado de NC Y UDP es beneficioso.

# 7
# Bibliografía


[1] D. Gómez, S. Hassayoun, A. Herrero, R. Agüero, D. Ros and M. García-Arranz. On the Addition of a Network Coding Layer within an Open Connectivity Services Framework, 2013

[2] 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.

[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, june 2008.

[4] Susumu HORIGUCHI Kaikai CHI, Xiaohong JIANG. A more efficent cope architecture for network coding in multihop wireless networks. IEICE TRANS. COMMUN., VOL.E92-B, NO 3, 10:766–776, 2009.

[5] Don Towsley Yong Huang, Majid Ghaderi and Weibo Gong. Tcp performance in coded wireless mesh networks. In IEEE SECON 2008, volume 9, 2008.

[6] S. Hassayoun, P. Maille, and D. Ros. On the impact of random losses on tcp performance in coded wireless mesh networks. In INFOCOM, 2010 Proceedings IEEE, pages 1 –9, march 2010.

[7] R. Koetter and M. Medard. An algebraic approach to network coding. In Information Theory, 2001. Proceedings. 2001 IEEE International Symposium on, page 104, 2001.

[8] Ralf Koetter David R. Karger Michelle Effros Jun Shi Tracey Ho, Muriel Médard and Ben Leong. A random linear network coding approach to multicast. IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 52 NO. 10, 18:4413–4430, 2006.

[9] Raymond W. Yeung. Avalanche: A network coding analysis. Commun. Inf. Syst. Volume 7, Number 4, 5:353–358, 2007.

[10] Albert S. J. Helberg Suné von Solms. The implementation of avalanche decoding in random network coding network. page 6.

[11] Network Simulator 3 `http://www.nsnam.org/overview/what-is-ns-3`