Índice

1. Introducción **1**

1.1. Planteamiento del problema 1

1.2. Objetivos 2

1.3. Estructura de la memoria 2

2. Estado del arte **4**

2.1. Redes Inalámbricas Malladas 4

2.2. Network Coding 6

2.2.1. Posibles problemas de network-coding 11

2.3. More 12

2.4. Antecedentes 16

3. Network Simulator **18**

3.1. Características 18

3.2. Componentes 20

3.2.1. Aplicación 20

3.2.2. UDP 21

3.3. Network Coding en NS-3 23

4. Desarrolo en el marco de NS-3 **26**

4.1. Módulos empleados en NS-3 26

4.1.1. Cabecera Network Coding 26

4.1.2. Proceso de codificación 29

4.1.3. Proceso de decodificación 30

4.2. Archivos de configuración del canal 32
4.3. Ficheros traza de salida 33
4.4. Automatización de la simulación 36

5. Simulaciones y resultados **38**
5.1. Conceptos fundamentales 38
5.2. Proceso de medida 39
5.3. Escenario estudiado 40
5.3.1. Resultados de las simulaciones 41

6. Conclusiones y líneas futuras **51**
6.1. Conclusiones y líneas futuras 51

Lista de acrónimos **53**

Índice de Figuras

2.1. Topología red mallada inalámbrica 6

2.2. Ejemplo de información tratada por nodo intermedio 7

2.3. Topología con nodos intermedios codificando paquetes 8

2.4. Ejemplo COPE sin codificación 9

2.5. Ejemplo COPE con codificación 9

2.6. Red en mariposa 10

2.7. Caso 1 y Caso 2 10

2.8. Caso 3 11

2.9. Codificación en la fuente 12

2.10. Pila de protocolos con MORE 13

2.11. Ejemplo unicast 14

2.12. Ejemplo multicast 14

3.1. Cronología Network Simulator 19

3.2. Organización de NS-3 20

3.3. Flujo de un paquete en NS-3 23

4.1. Cabecera *network-coding* 28

4.2. Parte variable cabecera 28

4.3. Proceso de transmisión 31

4.4. Fichero de configuración 32

4.5. Fichero localización física de los nodos 32

4.6. Fichero de configuración del canal 33

4.7. Fichero de configuración de encaminamiento 33

4.8. Formato de fichero 35
4.9. Formato de fichero 36
4.10. Script para simulaciones 37

5.1. Escenario simulado 40
5.2. Función distribución del throughput para la cabecera inicial 42
5.3. Función distribución del throughput para la cabecera final 43
5.4. Variación del throughput en función de K 44
5.5. Transmisiones innecesarias 45
5.6. Tiempos de rango y de matriz inversa 46
5.7. Variación del retardo y los datos subidos a aplicación 47
5.8. Throughput en un canal con errores 48
5.9. Retransmisiones a nivel físico 49

Resumen ejecutivo

Durante los últimos años se ha producido un crecimiento exponencial en cuanto a lo que se refiere a las necesidades y exigencias del usuario a la hora de utilizar redes inalámbricas. Esto ha supuesto que vayan surgiendo multitud de técnicas y mecanismos que tratan de mejorar el rendimiento de dichas redes, tratando de acercarse a las prestaciones ofrecidas por las redes cableadas.

A lo largo de este documento se presenta un estudio de la influencia que tienen las técnicas de codificación sobre redes inalámbricas, con el fin de conseguir mejoras en el rendimiento, en términos de throughput. En particular se analizará el mecanismo MORE sobre tráfico UDP, con el fin de cuantificar y valorar los beneficios obtenidos mediante la codificación aleatoria de la información por parte del nodo transmisor. Se realizará una comparación entre dichos resultados y los conseguidos con el protocolo TCP.

Para analizar los beneficios y posibles inconvenientes de la aplicación de esta técnica, en este proyecto se ha hecho uso de la herramienta NS-3. Tras la implementación de la nueva entidad (MORE), y tras realizar un gran número de simulaciones, se ha observado un incremento del rendimiento respecto a una solución más tradicional, basada en el protocolo TCP. Es importante destacar que, a pesar de que existen diferentes estudios similares, han sido pocos los que se han centrado en redes inalámbricas, ya que la mayoría son trabajos llevados a cabo sobre redes cableadas.

Abstract

In the latest years, the needs and requests from the users of wireless networks have experienced an exponential growth. This has brought about the appearance of several new techniques , aimed at improving the performance of such networks, trying to get closer to the performances offered by wired networks.

This project studies the influence that coding techniques might have over wireless networks, in order to improve their performance. In particular, we carry out an analysis of the MORE mechanism over the UDP protocol, assesing the benefits of using random coding at transmitter. A comparison between such results and the ones obtained using the legacy TCP protocol will be also presented in this document.

In order to analyze the benefits and potential drawbacks of this technique, we have used the NS-3 tool. After developing the new entity (MORE) and performing a large number of simulations, we have seen observed a performance improvement in comparison with a more traditional solution based on the TCP protocol. It is worth mentioning that, although there ara several similar studies, those have focused on wired networks.

Agradecimientos

Quiero expresar mi agradecimiento a mi tutor, Ramón Agüero, que me ha dado la gran oportunidad de realizar este proyecto con este gran grupo, depositando en mí su confianza y apoyo.

En segundo lugar agradezco enormemente el apoyo y aguante que ha tenido David Gómez durante todo este proceso. Gracias por la gran ayuda que me has dado.

Gracias a mis padres, que cada uno a su manera han estado atentos durante todo el trabajo. A mi madre que es la que lo ha vivido día a día y a mi padre, que entre partidos y cenas siempre teníamos algún momento para hablar del proyecto. A mi hermana, que a pesar de haberlo vivido en la distancia, también tuvo tiempo para preocuparse, ya que si soy teleco es en parte por ella y mi tío Klaus.

Al resto de mi familia, especialmente a mis abuelas, y a mis abuelos, que estoy seguro de que estarían muy orgullosos.

Por último no quiero olvidarme de mis amigos, que han conseguido que me dé realmente pena dejar la universidad, ya que sin lugar a dudas han sido los mejores años que he vivido.

1

Introducción

En este primer capítulo se describe la motivación de este trabajo. Se explican los objetivos fundamentales de esta memoria para, finalmente, describir la estructura que sigue el documento.

1.1 Planteamiento del problema

A lo largo de este proyecto de investigación se va a estudiar la influencia que tienen las técnicas de *network-coding* sobre redes malladas inalámbricas (WMN). Las tecnologías inalámbricas han acaparado la atención de la sociedad, convirtiéndose en el segmento de mayor y más rapido crecimiento en las telecomunicaciones. Su objetivo principal ha sido dotar de movilidad a los nodos de la red, beneficiándose de la ausencia de cableado. En los útimos años, las tecnologías de comunicación inalámbrica han llegado a un punto en el que su despliegue es tal, que aparecen en todos aquellos entornos en los que no hay infraestructuras de comunicación, en la mayoría de los casos debido al gran esfuerzo económico y dificultad que supone acometer dicho despliegue. Como es lógico, ante la falta de dicha infraestructura, para llevar a cabo la transmisión de informacion no hay otra opción que depender de las redes inalámbricas; esto conlleva una serie de problemas que presenta el uso de esta tecnología, que no son otros que:

- La hostilidad del canal, ya que se transmite a través del aire, lo que da lugar a la aparición de errores, suponiendo en muchos casos la pérdida de información.
- La seguridad a la hora de transmitir la información. Al utilizar el aire como medio de transmisión, la información puede ser captada por otros dispositivos.
- Las interferencias, ya que en función de la frecuencia utilizada por una red, esta puede verse afectada por la utilización de otros dispositivos.

Estos problemas suponen que el rendimiento de los sistemas inlalámbricos se vea claramente penalizado. Es por esto que con la introducción de nuevas técnicas de codificación, se va a tratar de reducir y minimizar algunos de los problemas que plantea esta tecnología, y así conseguir una mejora en el rendimiento de la red. El uso de esquemas de codificación también tiene como consecuencia que sea menos vulnerable ante posibles intrusos. Para alcanzar dicho objetivo se ha trabajado con el protocolo de transporte UDP, que no incorpora ningún tipo de control de congestión, ya que es no orientado a conexión.

El lenguaje de programación elegido para todas las implementaciones llevadas a cabo en este proyecto es C++, por razones de comodidad, y eficiencia computacional. Su naturaleza orientada a objetos y la existencia de múltiples tipos de estructuras de datos, permite un desarrollo modular de las diferentes clases y funciones implementadas.

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 codificación sobre tráfico UDP en redes inalámbricas. Para este cometido se ha utilizado como herramienta para desarrollar las simulaciones Network Simulator 3. Para analizar la mejora proporcionada por las técnicas propuestas se hará uso de una topología sencilla.

Se lleva a cabo un análisis exhaustivo de la influencia de diferentes parámetros operacionales del esquema de codificación, como son el número de paquetes que se pueden almacenar, el retardo en recepción, los tiempos empleados en la realización de ciertas operaciones necesarias para la codificación, los errores del canal, etc.

Con todo este análisis se obtienen unos resultados que permiten constatar que el uso de la codificación en redes inalámbricas supone un claro beneficio en cuanto al rendimiento de la red. Como se ha mencionado anteriormente se usará el simulador NS3, ya que incluye ciertos módulos que facilitan notablemente el trabajo a la hora de implementar tanto el proceso de codificación como de decodificación.

1.3 Estructura de la memoria

El documento se encuentra estructurado en los capítulos que se detallan a continuación:

- En el Capítulo 2 se realiza un acercamiento a los diferentes aspectos teóricos fundamentales que sirven de apoyo para comprender adecuadamente los temas tratados

posteriormente. Entre los aspectos principales se destaca las WMN y la repercusión de las técnicas de codificación en las mismas, para finalizar con la semilla fundamental del trabajo, el protocolo MORE. También se describirán los estudios que se han ido realizando sobre estas técnicas, para dar una idea de los puntos en los que la investigación se ha ido centrando hasta la actualidad. La intención es proporcionar una base sólida teórica acerca de las herramientas y técnicas utilizadas en este trabajo.

- El Capítulo 3 introduce la herramienta de simulación utilizada durante todo el proyecto. Network Simulator 3 es el entorno que ha permitido realizar las simulaciones necesarias para obtener los resultados finales. En primer lugar se detalla su estructura, características y los módulos más utilizados, para terminar con la explicación detallada de aquellos que ha sido necesario implementar para realizar las simulaciones.

- El Capítulo 4 nos profundiza en el funcionamiento de la codificación en el simulador, explicando los procesos de codificación y decodificación, la cabecera de la nueva entidad, etc. También se explicará el formato de los ficheros de trazas y el script necesario para automatizar las simulaciones.

- En el Capítulo 5 se presentan los parámetros que se han recogido a lo largo de todas las simulaciones realizadas y se describen los resultaos obtenidos. Se explicará asimismo la topología empleada.

- En el Capítulo 6 se sintetizan los resultados más relevantes del trabajo tras los análisis que se han llevado a cabo en el capítulo anterior. Se proponen además las posibles líneas de investigación que quedan abiertas para el futuro.

2

Estado del arte

En este segundo capítulo se realiza una introducción a los principales aspectos que se van a analizar y utilizar a lo largo de todo el trabajo, para que de esta manera el lector sea capaz de poder comprender con mejor claridad los distintos puntos que van a ser tratados en las siguientes páginas. Se hace especial hincapié en el *network-coding* y en MORE.

2.1 Redes Inalámbricas Malladas

En la actualidad la importancia de las redes malladas inalámbricas es cada vez mayor, ya que permiten proporcionar acceso barato a internet desde cualquier parte. Sin embargo, en muchas ocasiones sufren una mala calidad, causada por las estructuras urbanas e incluso por las interferencias producidas por otras tecnologías. Además, compiten con otras tecnologías como cable, internet vía satélite, xDSL.

Su estructura es similar a la de las redes ad-hoc, en las que cada usuario no solo hace las funciones de transmisor y receptor, sino que también actúa como un router dentro de la red. La mayor diferencia entre ambas se encuentra en el tipo de tráfico que hay en la red, mientras que en redes malladas todo el tráfico está habitualmente dirigido desde o hacia una puerta de salida a internet, en las redes ad-hoc el tráfico suele afectar a pares de nodos cualesquiera de la red.

En definitiva, cuando se habla de redes inalámbricas malladas, se suele referir a aquellas en las que existen uno o más nodos intermedios que se encargan de reenviar y recibir los paquetes para facilitar la comunicación entre nodos que no se pueden comunicar de forma directa.

En definitiva, las redes malladas permiten:

- Ampliar la cobertura de la red

- Mejorar la conectividad
- Mejorar la escalabilidad
- Aumentar la velocidad de transmisión

Y presentan una serie de ventajas adicionales:

- Reducido coste de instalación y mantenimiento, en comparación con las redes cableadas
- Despliegue rápido
- Gran facilidad para ofrecer cobertura en zonas de difícil acceso para tecnologías cableadas
- Aumentar la velocidad de transmisión

Todo esto tiene como resultado un incremento del rendimiento y de la eficiencia de la red. Cada nodo posee un rango de cobertura inalámbrica específico, y para que pueda existir un enlace entre dos nodos, ambos deben estar dentro del radio de cobertura del otro.

Las técnicas de enrutamiento que utilizan este tipo de redes, al igual que las cableadas, intentan elegir la mejor secuencia de nodos intermedios entre el origen y el destino para transmitir la información. Cuando se pretende que un nuevo nodo entre a formar parte de la red, este lo hace de forma dinámica y autónoma. Por otro lado, una de sus principales desventajas es la capacidad que puede ofrecer, ya que conforme va aumentando el tamaño de la red sube número de saltos necesarios para alcanzar el destino, así como las interferencias que se producen entre ellos.

Existen estudios que consideran un problema el uso del protocolo TCP en redes inalámbricas, ya que fue diseñado para redes cableadas, en las que la probabilidad de error era casi despreciable. A pesar de que se han ido realizando diversas modificaciones de TCP para intentar minimizar la pérdida de rendimiento, todavía no se ha encontrado una solución óptima.

En la Figura 2.1 se muestra un ejemplo de red mallada inalámbrica.

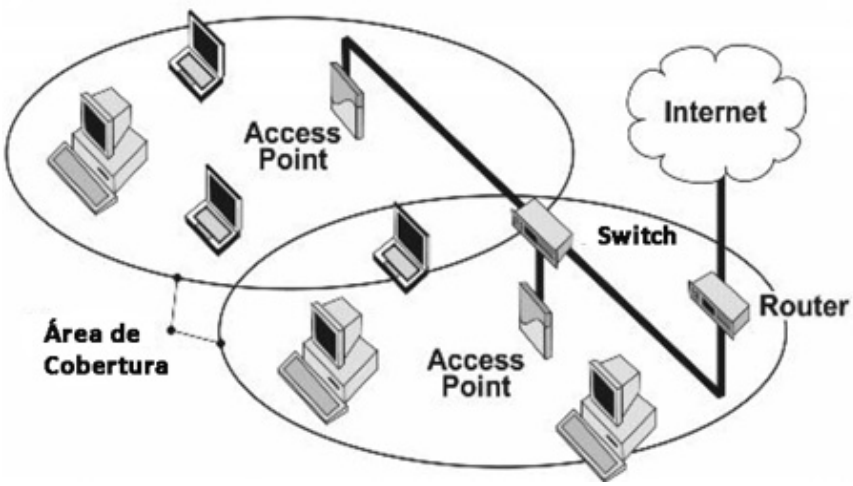


Figura 2.1: Topología red mallada inalámbrica

2.2 Network Coding

En primer lugar se describe la técnica principal sobre la que gira este trabajo, el *network-coding*. Se espera que sea muy utilizada por las redes actuales y futuras, ya que pone gran cantidad de beneficios al servicio del usuario.

Sus orígenes datan de finales de los años noventa, y hay que atribuirselos al matemático alemán Rudolf F. Ahlswede, quien se encargó de demostrar que podía aportar unas mejoras fundamentales[1]:

- Se podía mejorar el rendimiento de la red
- Se conseguía reducir los retardos
- Se obtenía una red mucho más robusta y segura

A continuación se explicarán los principios básicos del *network-coding* para finalmente detallar el protocolo utilizado en el marco de este proyecto, MORE. Hasta hace algunos años, lo normal era que la información viajara a través de las redes de comunicación sin sufrir ningún tipo de manipulación; es decir, los paquetes que salían del origen y llegaban al destino, no habían sufrido ningún cambio salvo los necesarios en las cabeceras correspondientes. Con el *network-coding* todo esto va a cambiar, ya que la información que viaja a través de las redes de comunicación va a comenzar a ser tratada, tal y como se muestra en la Figura 2.2.

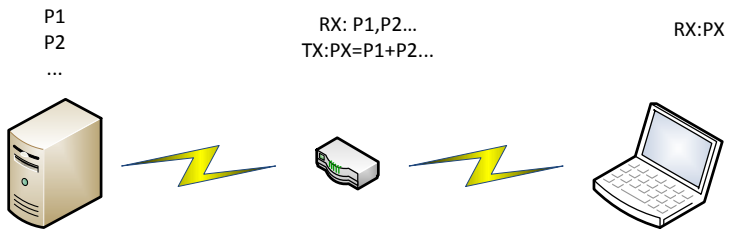


Figura 2.2: Ejemplo de información tratada por nodo intermedio

La base o característica principal de esta innovadora técnica es, como ya he mencionado anteriormente, que la información va a ser modificada, y esto es posible gracias a los avances tecnológicos que hoy en día se han producido en las técnicas de codificación. Hasta ahora, en las redes tradicionales, los nodos intermedios no actuaban más que como simples retransmisores de la información que les iba llegando; por lo tanto, si un nodo recibía dos paquetes de dos fuentes, aunque ambos se dirigieran al mismo destino, no se transmitirían a la vez, provocando la espera en la cola de transmisión; esto supone envíos separados para cada paquete, lo cual disminuye la eficiencia de la red.

Con el *network-coding*, se emplean algoritmos para cambiar esos dos mensajes, enviando únicamente este último. Con esta nueva técnica, los nodos serán capaces de procesar los paquetes que les llegan, dejando de esta forma de recibir y reenviar la información hacia el siguiente nodo. Esto implica que se requiere un menor número de transmisiones para enviar todos los datos, a expensas de un mayor procesado por parte de los nodos intermedios y el destino. Todo esto supone un gran cambio y giro innovador respecto al envío de información a través de redes inalámbricas de comunicación.

Con todo esto se llegó a la conclusión de que era posible que nodos intermedios realizaran operaciones sobre flujos independientes de datos, procesándolos y transmitiéndolos conjuntamente, como se puede ver en la Figura 2.3. Así, se consigue mejorar la capacidad de transmisiones multicast en una red cuando los nodos intermedios (routers) se encargan de mezclar la información de varios mensajes en un único paquete. En el momento en el que el receptor recibe el paquete, obtendrá la información que estaba destinada a él si y solo si tiene la información necesaria para decodificar el paquete recibido.

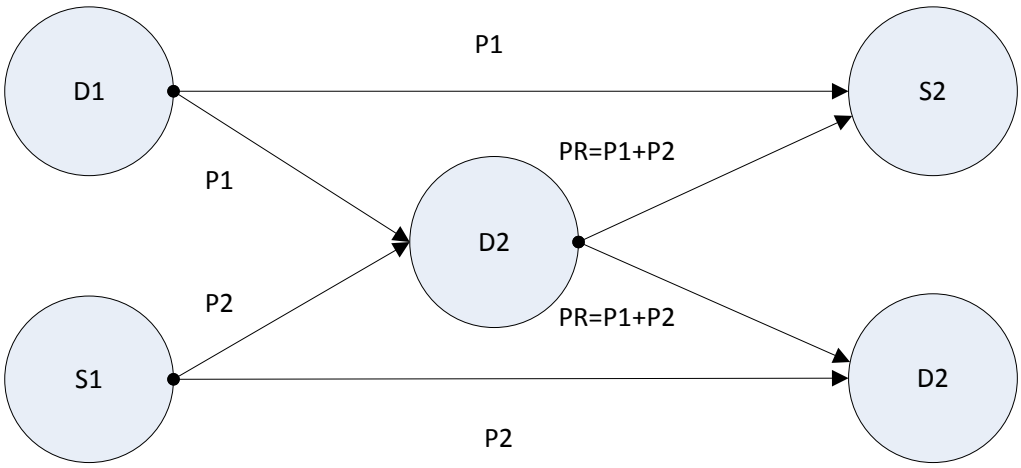


Figura 2.3: Topología con nodos intermedios codificando paquetes

Diversos investigadores llevaron a cabo el uso de esta técnica sobre sistemas inalámbricos, y demostraron que se conseguía minimizar los costes de comunicación. En el 2006 se presentó la propuesta para redes inalámbricas conocida como *COPE*[2]; consiste en una nueva arquitectura para redes inalámbricas malladas, que introduce una entidad codificadora entre las capas IP y MAC, lo que da como resultado una mejora sustancial del rendimiento. Esta capa que se introduce, se encarga de identificar las diferentes oportunidades de codificación y, a continuación, transmitir múltiples paquetes de datos codificados en un envío único. Desde su aparición, el interés despertado por esta técnica ha sido notable. Aunque principalmente esta técnica está pensada para mejorar el rendimiento en redes donde diferentes usuarios comparten recursos, se ha trasladado a otros muchos ámbitos, como pueden ser la monitorización de redes, el almacenamiento distribuido o la seguridad.

Como ya se ha mencionado, el diseño de COPE se inspira en la codificación de red, y se va a utilizar un simple ejemplo para comprenderlo mejor. Se considera la topología de la Figura 2.4, donde los nodos A y B quieren intercambiar dos paquetes a través de un router. Haciendo uso de las técnicas tradicionales, el equipo A envía su paquete al router, que lo envía posteriormente al equipo B; por su parte, B mandará su paquete al router, que lo reenviará al equipo A. En total han sido necesarias cuatro transmisiones durante todo el proceso.

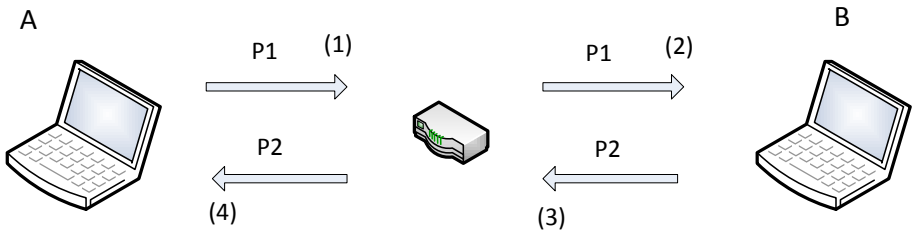


Figura 2.4: Ejemplo COPE sin codificación

Una vez visto el ejemplo anterior, ahora se va a considerar un nuevo enfoque, en el que los equipos A y B envían respectivamente sus paquetes al router, que se encarga de realizar la operación XOR sobre los mismos, dando lugar a un único paquete que transmite (de manera broadcast) hacia ambos destinos, como se ve en la Figura 2.5. Tanto el A como B pueden recuperar los paquetes, ya que ambos tienen los que ellos transmitieron. Se ha ahorrado una transmisión, con el consiguiente aumento de rendimiento de la red.

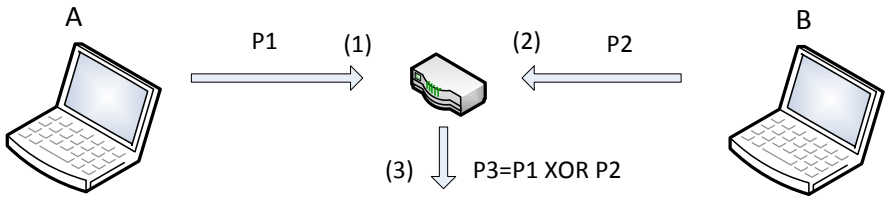


Figura 2.5: Ejemplo COPE con codificación

En el caso anterior se ha empleado una topología de red muy sencilla. En la vida real no siempre serán tan simples, por lo que se debería generalizar la solución propuesta. Una de las topologías más utilizadas para explicar esta técnica es la conocida como la mariposa. Se observa la topología correspondiente en la Figura 2.6.

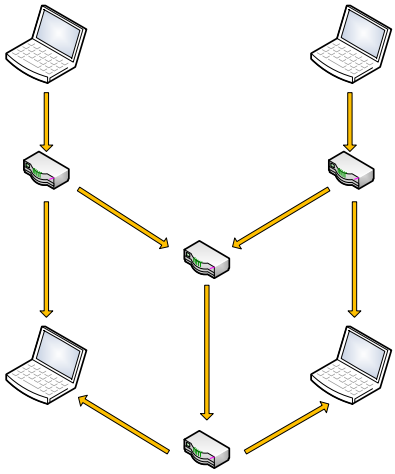


Figura 2.6: Red en mariposa

Se plantean varias posibilidades; en primer lugar, se considera que el nodo D1 es el receptor de todos los envíos (Figura 2.7), por lo que el paquete enviado por S2 seguirá el camino S2 $\longrightarrow$ R2,R2$\longrightarrow$R3,R3$\longrightarrow$R4 y R4$\longrightarrow$D1, mientras que el enviado por S1 realizará el camino S1$\longrightarrow$R1 y R1$\longrightarrow$D1. Después se analizará una situación similar (Figura 2.7), en la que el receptor de todos los paquetes es el nodo D2. Se da por hecho que todos los enlaces tienen la capacidad de enviar un único paquete por unidad de tiempo.

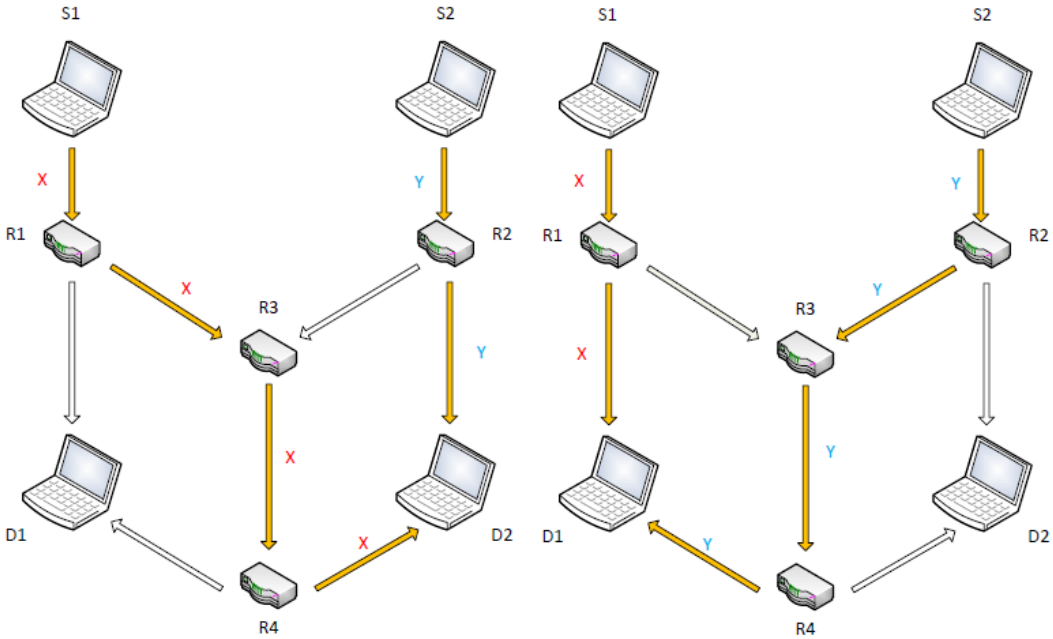


Figura 2.7: Caso 1 y Caso 2

El ejemplo que resulta más interesante es el que muestra la Figura 2.8, donde el nodo S1 pretende enviar el paquete P1 al nodo D2, mientras que S2 quiere enviar el paquete P2 al nodo D1. Si se supone que los nodos intermedios R3 y R4 se limitan a reenviar los paquetes que les van llegando, se crearía un cuello de botella en el enlace entre ambos, teniendo que transmitir uno detrás del otro. La solución a este problema es sencilla; el primero de los nodos intermedios realiza una codificación de ambos paquetes mediante la operación XOR y se envía el paquete codificado (P1⊕P2), que recibirán ambos nodos receptores. Dado que los nodos receptores ya han recibido también los respectivos paquetes necesarios para decodificar, no tendrán más que realizar la operación pertinente y así conseguir los datos que iban destinados hacia él. Este es un ejemplo más que pone de manifiesto que mediante esta técnica se puede aumentar de manera considerable el rendimiento, minimizando el número de transmisiones necesarias para transportar diferentes flujos de información en la red, dado que los datos de varios paquetes se combinarán en uno solo.

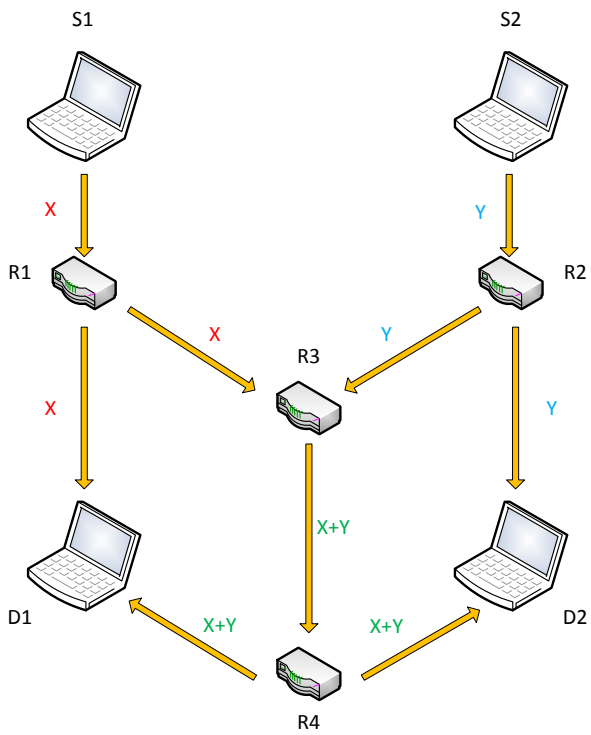


Figura 2.8: Caso 3

2.2.1. Posibles problemas de network-coding

Aunque a primera vista parece que todo son beneficios y mejoras con el *network-coding*, no todo es tan sencillo, ya que su implementación acarrea una serie de problemas,

entre los que se destaca el de las retransmisiones del nivel MAC de 802.11. Cuando se realiza la transmisión de un paquete codificado hacia varios receptores, se hace con una modalidad modificada de broadcast, y este tipo de envíos tienen un problema, ya que una vez enviado el paquete no puede ser retransmitido por la capa MAC, con lo que se perdería fiabilidad en la transmisión. Si se realizara un envío unicast, se solucionaría el problema de las retransmisiones, aportando una mayor fiabilidad en casos en los que el canal no sea bueno, pero también presenta un problema y es que el paquete puede ir dirigido únicamente a una dirección MAC. La solución es el empleo de un envío mezcla entre unicast y broadcast, en el que todos los nodos pueden recibir y procesar los paquetes (over hearing).

2.3 More

Como ya se ha adelantado, el trabajo se iba a centrar principalmente en una manera alternativa de *network-coding*, MORE[3]. Se trata de un nuevo protocolo de enrutamiento oportunista para redes inalámbricas que, además, evita la necesidad de disponer de un nodo de coordinación y está apoyado lógicamente sobre los pilares de *network-coding*. En este caso, los paquetes que se mezclan al azar para posteriormente ser codificados, van todos al mismo destino, que se encargará de decodificarlos para recuperar los paquetes originales. Este nuevo enfoque no requiere coordinación, por lo que aporta una serie de ventajas para mejorar el rendimiento.

MORE surge inicialmente como alternativa a la complejidad y problemática que requería el protocolo ExOR[4]. La novedad principal de este nuevo protocolo es que mezcla, o codifica, los paquetes antes de enviarlos como se ve en la Figura 2.9.

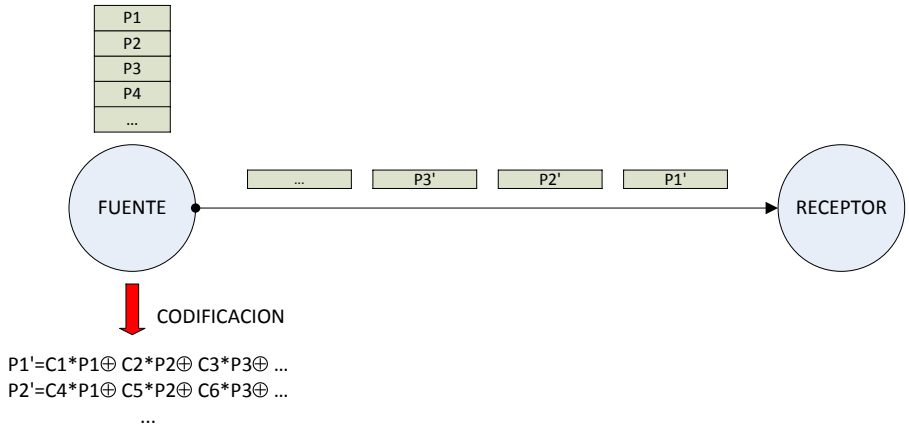


Figura 2.9: Codificación en la fuente

Así se garantiza que los routers que escuchan las mismas transmisiones no reenvíen paquetes innecesarios. Además es poco probable que distintos paquetes, codificados al azar, sean iguales y, obviamente, dicha probabilidad va disminuyendo conforme aumenta el número de paquetes codificados. Como resultado, MORE no requiere de una estructura compleja, por lo que trabaja directamente entre las capas UDP e IP, como muestra la Figura 2.10.

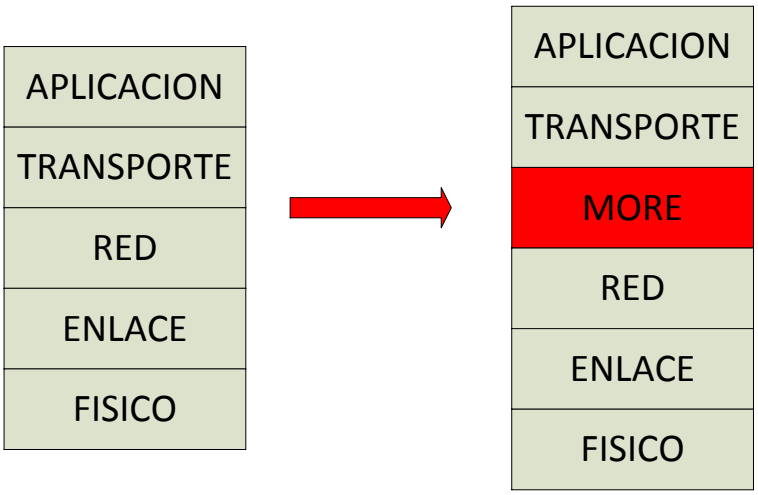


Figura 2.10: Pila de protocolos con MORE

Como se ha mencionado, el diseño de MORE se basa en la teoría de la codificación de red, y para tratar de comprender un poco mejor su funcionamiento se van a plantear dos ejemplos simples, uno con tráfico unicast, y otro con multicast:

- Unicast. El escenario es el que se ve en la Figura 2.11; bajo un esquema de enrutamiento tradicional, el camino se establecería antes de realizar la transmisión. Se realizaría el envío de la información a través de la ruta src$\longrightarrow$R$\longrightarrow$dest, que presenta una fiabilidad mayor. Sin embargo, al trabajar en el medio inalámbrico, puede suceder que cuando un nodo realice la transmisión de un paquete, existe la posibilidad de que un nodo diferente al elegido como próximo salto escuchade la transmisión. Por ejemplo supóngase que el origen (Src) envía dos paquetes (P_1 yP_2), y se da el caso de que el próximo salto, en este caso R, recibe correctamente ambos, pero el destino también ha escuchado una de las transmisiones (P_2). Parecería que sería innecesario volver a transmitir P_2 y, por esto, EXoR requiere de un nodo coordinador, lo cual es bastante complejo. Volviendo al ejemplo, R debería transmitir solamente el paquete P_1, porque el destino (Dest) ya ha recibido P_2, pero R no tiene manera de saber qué paquete debe transmitir y cual no. Este problema se hace más complicado conforme

va aumentando el tamaño de las redes. Ante esta circunstancia, se plantea una solución basada en codificación; en el ejemplo que se está usando el destino ha escuchado el paquete P_2 y el nodo R no es consciente, como ya se ha mencionado, la circunstancia no origina mayor sobrecarga, ya que el nodo R puede enviar un único paquete, como por ejemplo la suma de ambos paquetes $(P_1 + P_2)$, y el destino será capaz de recuperar P_1, simplemente realizando la resta de ambos. De esta forma, R no necesita saber que paquetes ha recibido con anterioridad el destino. Si se generaliza, se puede decir que el nodo R se encarga de realizar combinaciones lineales aleatorias de los paquetes que escucha. Una vez recibida toda la información necesaria, el destino enviará un ACK para confirmar que ha recibido toda la información correctamente. Este enfoque no requiere de nodo de coordinación.

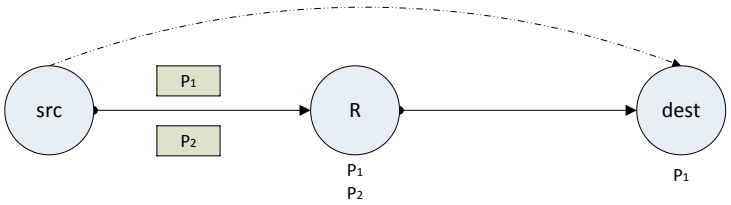


Figura 2.11: Ejemplo unicast

- Multicast. En este segundo ejemplo se destaca la relación que existe entre el *network-coding* y el multicast. En la Figura 2.12, se ve una fuente que envía cuatro paquetes (P_1,P_2,P_3 y P_4) a tres destinos. Supóngase que cada destino recibe únicamente los paquetes indicados en la figura, es decir, el primer destino recibe P_1 y P_2, el segundo destino recibe P_2 y P_3, y el último destino recibe P_3 y P_4.

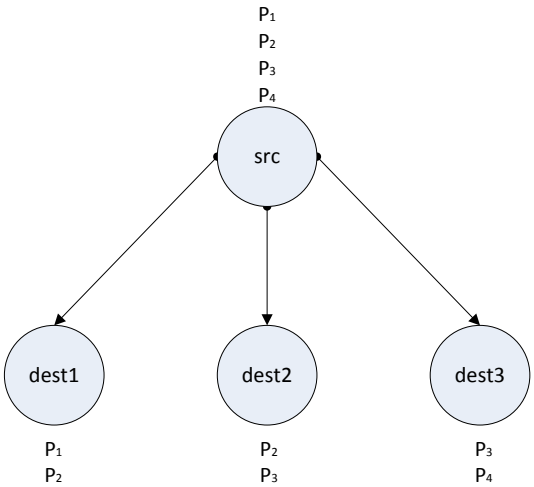


Figura 2.12: Ejemplo multicast

Si no se hace uso de ninguna codificación, la fuente tendría que retransmitir otra vez la totalidad de los cuatro paquetes perdidos, sin embargo, si la fuente codificara los paquetes que transmite, sería suficiente con transmitir dos únicos paquetes codificados para que los tres destinos puedan obtener los paquetes correspondientes. Si se envía un paquete $P_1^* = P_1+P_2+P_3+P_4$ y otro $P_2^* = P_1+2P_2+3P_3+4P_4$; a pesar de que se perdieron varios paquetes, los tres destinos puedes ser capaces de recuperar los cuatro originales gracias al envío de estos dos nuevos paquetes codificados. Por ejemplo, el primer destino, que ha recibido P_1,P_2, P_1^* yP_2^*, puede recuperar los originales con la simple operación de invertir la matriz de coeficientes y multiplicarla por los recibidos, como se muestra a continuación.

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} P_1^* \\ P_2^* \\ P_1 \\ P_2 \end{bmatrix}$$

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} P_1^* \\ P_2^* \\ P_2 \\ P_3 \end{bmatrix}$$

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} P_1^* \\ P_2^* \\ P_3 \\ P_4 \end{bmatrix}$$

En este sencillo ejemplo se ha podido ver como haciendo uso de esta nueva técnica, la red ha conseguido reducir las retransmisiones necesarias, mejorando de esta forma el rendimiento y la eficiencia de la red.

Para llevar a cabo el diseño de un protocolo de estas características, es necesario hacer frente a una serie de problemas iniciales, que más adelante se describirán, algunos de estos problemas son los siguientes:

- ¿ Cuántas transmisiones se deben realizar? En el enrutamiento tradicional un nodo sigue enviando un paquete hasta que el siguiente lo recibe o hasta que desiste, pasado una serie de intentos. Con *network-coding*, al no existir un próximo salto en particular, todos los nodos cercanos al destino pueden participar en la transmisión del paquete. Es más complicado establecer el número de transmisiones para garantizar la correcta recepción del paquete.

- ¿Es realmente eficiente en cuanto a su complejidad? El uso de esta técnica optimiza sin lugar a duda la utilización del medio inalámbrico, pero a su vez requiere que los nodos sean capaces de llevar a cabo una serie de operaciones. Esto supone que serán necesarias estrategias eficaces de codificación y decodificación para evitar las posibles creaciones de cuellos de botella.

A continuación se van a explicar una serie de términos que son esenciales y serán empleados a lo largo del proyecto:

- Paquete nativo: se refiere a un paquete que aún no ha sido codificado, esto es, un paquete original.
- Vector de coeficientes: será un vector aleatorio compuesto únicamente por 1 ó 0, ya que se ha trabajado con el cuerpo finito GF(2), que servirá para conocer como han sido codificados los paquetes nativos para llegar al paquete codificado. De esta manera un paquete codificado consta del vector aleatorio y los paquetes nativos.
- Paquete exitoso: todo paquete que llegue a un nodo y sea linealmente independiente de los que han llegado anteriormente será considerado como válido y pasará a formar parte de la matriz, puesto que el hecho de que sea linealmente independiente de los demás indica que aporta información nueva.

2.4 Antecedentes

Durante los últimos años, se han realizado diferentes estudios acerca del *network-coding*, tomando como principal pionero a R. Ahlswede[5], que se encargó de demostrar que la posibilidad de codificar paquetes durante la comunicación, podría aumentar de manera considerable el rendimiento en las redes inalámbricas actuales. Entre los diversos estudios realizados, en primer lugar cabe destacar el momento en el cual se implementa por primera vez una nueva arquitectura de envío, conocida como COPE. Es uno de los primeros trabajos de investigación en el que se comienza a realizar un estudio más práctico y real, en el que se incluyen simulaciones de escenarios con hasta veinte nodos. Entre las distintas características que tiene esta arquitectura, destaca alguna interesante, como por ejemplo que los nodos envían periódicamente informes, conocidos como reception reports, los cuales se encargan de indicar los paquetes que han conseguido llevar a cabo el overhearing. Esto provoca que se produzca una cierta sobrecarga en la red, pero permite que los nodos intermedios sepan los paquetes que deben codificar y cuáles deben ser enviados tal cual los recibieron. Sin embargo, dado que estos informes anteriormente citados a veces son poco útiles, cuando hay mucha sobrecarga o escaso uso de la red, los nodos hacen uso de la métrica ETX[6], computando la probabilidad de envío entre cada enlace

y asignando diferentes pesos, que se tendrán en cuenta a la hora de codificar los paquetes. Esta arquitectura incluye un sistema de envío de ACKs asíncronos que se incluyen en la nueva cabecera creada, para minimizar los problemas que hay en 802.11 en cuanto a la confirmación de paquetes. A pesar de todo, no llega a ser considerado como un sistema plenamente efectivo.

Hasta la fecha, no se han realizado muchos más trabajos estudiando las técnicas de *network-coding*, los patrones de tráfico UDP y las redes inalámbricas, no obstante se podría nombrar alguno en combinacion con TCP como en realizado por Towsley[7], que sintetiza un protocolo similar a COPE, en el cual los nodos codifican en función de lo que han enviado y recibido anteriormente, sin realizar el proceso de *oportunistic listening*. Los paquetes llevan asociado un temporizador, cuyo valor al aumentar, incrementará las oportunidades de codificación. El estudio demuestra que se pueden conseguir incrementos en el throughput de entre un 20 % y un 70 %.

También hay otros estudios similares donde se presentan los resultados obtenidos con escenarios del tipo X y mariposa[8], en combinación con tráfico TCP. En este caso el medio inalámbrico se sustituye por un medio cableado al que se le añaden errores aleatorios. Nuevamente se vuelve a verificar lo que ya se esperaba, y es que el uso de esta técnica da como resultado la mejora del rendimiento hasta un cierto nivel de errores, a partir del cual el sistema tradicional es más eficiente.

3

Network Simulator

A lo largo de este capítulo se va a llevar a cabo la introducción a la herramienta principal sobre la cual se ha desarrollado este proyecto, es decir, el simulador NS-3. En segundo lugar se describirá la estructura interna de alguno de sus elementos, ya que, algunos han sido más utilizados que otros. Finalmente se acabará con los cambios que se han realizado en los distintos módulos que componen el simulador y otros que han sido implementados.

3.1 Características

Los comienzos del simulador datan del año 1988 (vease la Figura 3.1), cuando Srinivasan Keshav comenzó el desarrolo de un simulador de redes bajo el nombre REAL, el cual es el antecedente a NS. Para encontrar la primera versión de NS hay que remontarse a comienzos de los años noventa, cuando el Laboratorio Lawrence Berkeley llevó a cabo su desarrollo, escribiendo su núcleo en C++ con scripts Tcl para la descripción de los escenarios a simular, esta primera versión fue llamada NS-1. Durante el periodo de vida de esta primera versión se produjeron contribuciones de Sun Microsystems y UC Berkeley, entre otros. Es en 1996-97 cuando surgió la segunda versión del proyecto, conocida como NS-2, y fue McCanne el encargado de sustituir el uso de Tcl por OTcl, una versión orientada a objetos de Tcl, mientras que se mantuvo el núcleo en C++. Durante los años 2004-2005 se comenzó a trabajar en la última versión conocida que es la usada en este trabajo, NS-3, bajo el trabajo liderado por Tom Henderson de la Universidad de Washington. Se trata de un simulador totalmente nuevo, creado desde cero, ya que se abandonó por completo la compatibilidad NS-2, debido a la alta sobrecarga de mantenimiento que podría suponer. En junio de 2008 se lanzó la primera versión, NS-3.1, y se sigue en un constante proceso de renovación, encontrándose actualmente por la version NS-3.19, publicada a finales de 2013.

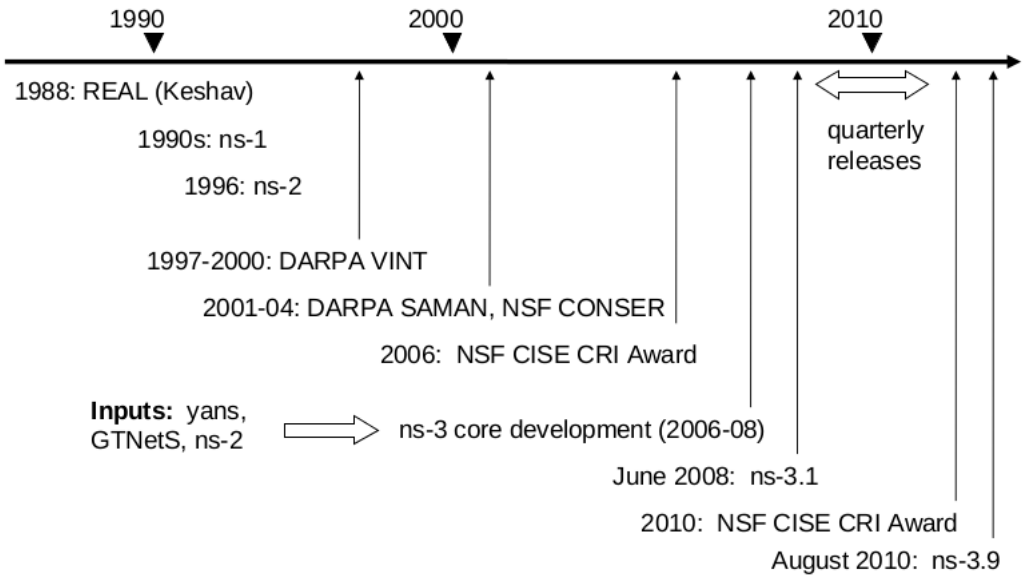


Figura 3.1: Cronología Network Simulator

Como ya se ha mencionado, ya se ha publicado la versión NS-3.19, pero en este proyecto se ha hecho uso de NS-3.13[9], que es de diciembre de 2011. El principal motivo es la reutilización de ciertos módulos llevados a cabo previemente sobre esta versión, además de la estabilidad que ofrece. De todas formas, las mejoras que ofrecen las sucesivas actualizaciones son escasas, siendo, en la mayoría de los casos, correcciones de pequeños errores y que poco o nada influyen en el desarrollo llevado a cabo en este trabajo.

Cabe destacar uno de los grandes atractivos de NS-3, y es su carácter de software libre bajo licencia GNU GPLV2, lo que ha facilitado su uso masivo en diferentes sectores, como son la industria, entornos académicos o, incluso, gobiernos. Resaltar también su flexibilidad y su carácter modular que, en labores de investigación, brinda la posibilidad de incorporar nuevos componentes con funcionalidades que anteriormente no estaban implementadas. Su uso es bastante complejo, puesto que requiere un conocimiento profundo de su arquitectura interna, pero las opciones de simulación que ofrece son muy amplias, permitiendo realizar estudios a gran escala, que hoy en día son esenciales.

NS-3 es un simulador de redes de eventos discretos, en el que tanto el núcleo como los modelos se implementan en C++. Para explicarlo de manera simple y sencilla, se cuenta con un programa C++ principal donde se define la topología de la simulación y enlaza con las diferentes librerías con las que cuenta NS-3, dando lugar al escenario que se desea simular. También se describen en el programa principal, las características básicas de la simulación, desde la cantidad de información transmitida al tiempo de la conexión. NS-3 tiene exportados la mayoría de sus API a Python, permitiendo crear escenarios de simulación también en este lenguaje.

El código fuente de NS-3 está mayormente organizado en el directorio /src y se estructura como se recoge 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 clara la estructura de cada uno de los módulos, para poder realizar un uso óptimo de la misma.

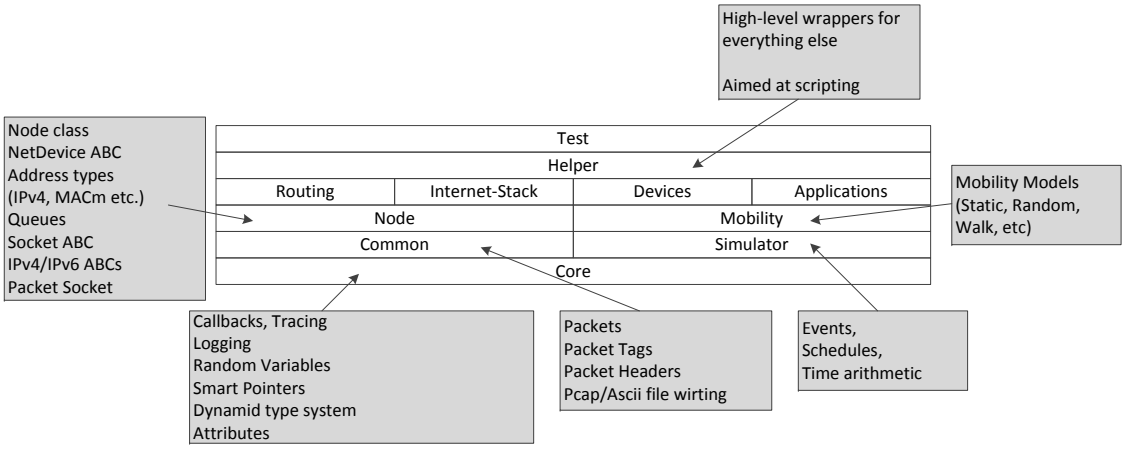


Figura 3.2: Organización de NS-3

La financiación inicial para llevar a cabo el desarrollo de NS-3 ha sido proporcionada por la fundación National Science y el grupo Planete en INRIA Sophia Antipolis. También se ha contado con la ayuda proporcionada por el departamento de Ingeniería Eléctrica y Computación del Instituto de Tecnología de Georgia y el Departamento de Ingeniería Eléctrica de la Universidad de Washington.

3.2 Componentes

El número de elementos que componen el simulador NS-3 son múltiples, cada uno de los cuales tiene a su vez varios ficheros de código C++, en los que se declaran diferentes clases, que dependen unas de otras. A continuación se describen los módulos más usados en este proyecto. En primer lugar se comentarán los principales componentes del nivel de aplicación, y después los de nivel de transporte.

3.2.1. Aplicación

En el nivel de aplicación se han utilizado dos aplicaciones muy básicas:

- OnOffApplication: se trata de una aplicación que se implementa en el transmisor, y genera tráfico dirigido a un único destino, siguiendo un patrón configurable. Permite

modificar varios parámetros que determinan el patrón de tráfico enviado. Los más destacables son OnTime, OffTime, DataRate, PacketSize y MaxBytes. Los dos primeros establecen la duración de los estados ON (transmitiendo) y OFF (apagado), mientras que DataRate define la velocidad de transmisión del nivel de aplicación hacia el nivel de transporte, y PacketSize determina el tamaño de los paquetes generados. Para terminar, la variable MaxBytes fija el número máximo de bytes enviados por la aplicación.

- PacketSink: se trata de una aplicación que se implementa en el nodo receptor y recibe tráfico dirigido a una dirección y puerto específicos. Fue diseñada para complementar a OnOffApplication. Sus parámetros más importantes son:
  - Local, que se encarga de asignar la dirección a través la que se va a recibir tráfico, y que se puede establecer para que sea cualquier dirección habilitada en el receptor.
  - StartTime y StopTime, que definen el intervalo temporal en el que la aplicación está activa para recibir tráfico.
  - Protocol, que determina el identificador correspondiente en el socket del receptor.

Estas son las dos aplicaciones utilizadas durante el estudio de los diferentes apartados que contiene esta memoria.

3.2.2. UDP

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: se encarga del establecimiento de los atributos del socket, que pueden ser usados a lo largo de las diversas implementaciones. Se encuentra definida en el fichero */src/internet/model/udp-socket(.cc/.h).*
- UdpSocketFactory: crea los sockets UDP. Define una interfaz de programación para los Sockets (API) y almacena las variables globales usadas por defecto a la hora de su inicialización. Se encuentra definida en el fichero */src/internet/model/ udp-socket-factory(.cc/.h).*
- UdpL4Protocol: esta clase se encarga de definir una capa intermedia entre el socket y el nivel 3, cuya función consiste en la interconexión entre estas capas, gracias a la asignación del *end point*, que se encarga de la correspondencia(*IpV4EndPoint*). Se encuentra definida en el fichero */src/internet/model/udp-l4-protocol(.cc/.h).*

- UdpSocketImpl: en esta clase se lleva a cabo la definición de los aspectos específicos del socket. Se encuentra definida en el fichero */src/internet/model/udp-socket-impl(.cc/.h).*
- IpV4EndPoint: se encarga del almacenamiento del puerto local, dirección local, puerto destino, y dirección destino, asociados al socket. Se encuentra definida en el fichero */src/internet/model/ipv4-end-point(.cc/.h).*
- UdpHeader: implementación de los campos que constituyen la cabecera UDP (número de puerto, datos, etc). Se encuentra definida en el fichero */src/internet/model/ udp-header(.cc/.h).*
- UdpSocketFactoryImpl: implementación de UdpSocketFactory para UDP nativo. Se encuentra definida en el fichero */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 permite la creación de módulos que sustituyen a los modelos nativos. El API pública de esos objetos en C++ se encuetra en el directorio */src/network/model*, donde destacan principalmente: *address.h*,*socket.h*,*node.h* y *packet.h*.

En la Figura 3.3 se representa el flujo de un paquete sobre nodos que tienen implementada la pila de protocolos UDP. Hay que tener en cuenta que es independiente de la capa física y de enlace de la red.

En primer lugar, el diagrama de la izquierda muestra el flujo que sigue un paquete transmitido. A nivel de aplicación se llama al socket para enviar el paquete, a través de la clase *UdpSocketImpl*. El socket se apoya en *UdpL4Protocol* para dirigirse al destino (en UDP no hay 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, donde se añadirá la cabecera enviándolo a *NetDevice*. En la transmisión hay un paso previo, el Address Resolution Protocol (ARP), que se recoge en el módulo *ArpIpv4Interface*, donde queda implementado el protocolo encargado de encontrar la dirección MAC del nodo al que desea transmitir.

En segundo lugar aparece el diagrama de la derecha, que representa el proceso contrario, es decir, cuando un paquete es recibido en el nodo, y llega hasta el 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, y 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.

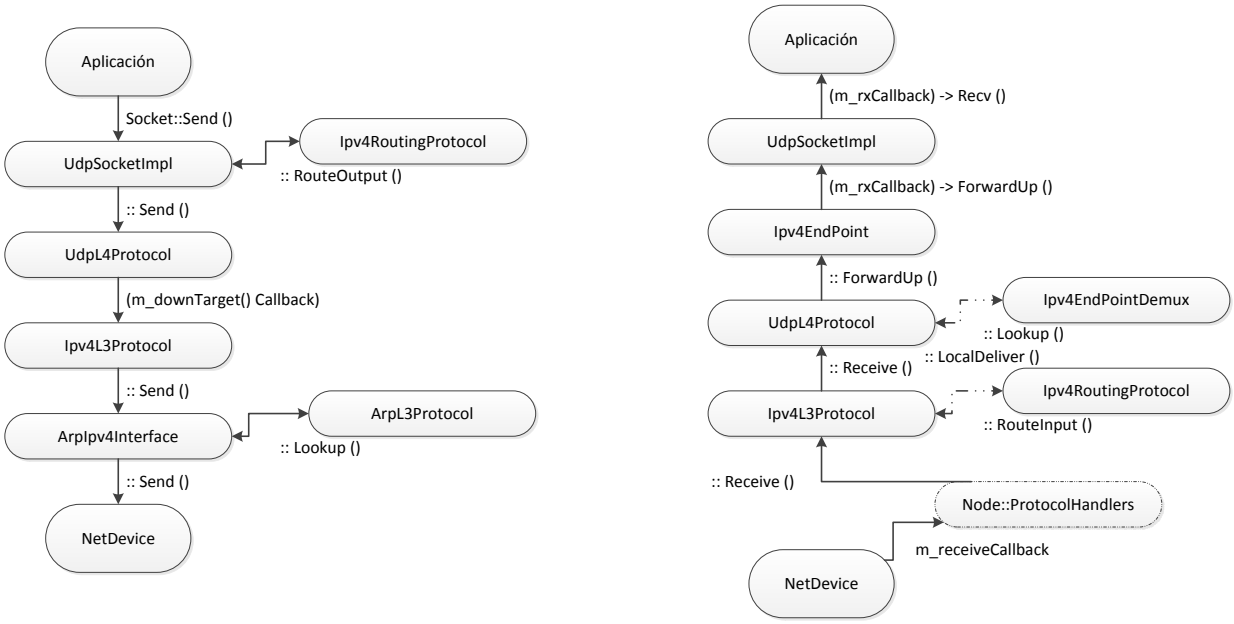


Figura 3.3: Flujo de un paquete en NS-3

3.3 Network Coding en NS-3

Puesto que el simulador no cuenta con ninguna implementación nativa del protocolo *network-coding*, es necesario añadir un nuevo módulo, que se sitúa en */src/network-coding*, y esta compuesto por los siguientes ficheros, cuyo desarrollo será explicado en el siguiente capítulo.

- MoreHeader: se utiliza para la implementación de la cabecera de *network-coding*, que es la nueva entidad entre la capa de transporte y la de red. Se encuentra en */src/network-coding/model/more-header(.cc/.h)*
- NetworkCodingL4Protocol: 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)*
- MoreNetworkCodingProtocol: consta de las principales funciones desempeñadas por los nodos que componen el escenario. Se encuentra en */src/network-coding/model/more-network-coding-protocol(.cc/.h)*. Se destacan las siguientes funciones:
    - ReceiveFromUpperLayer: en esta función se van recibiendo los paquetes desde la capa superior, esto es, la capa de transporte (UDP), y se irán almacenando

en un buffer de transmisión hasta un determinado momento, cuando se alcance un determinado valor de K paquetes. En ese momento se procederá a realizar la llamada a otra función básica y esencial de la implementación llevada a cabo, que será explicada con gran detalle en el próximo capítulo, la función *Codificar*, que, como su propio nombre indica, se encargará de la codificación de los paquetes. Cabe destacar que los paquetes que van llegando de UDP serán almacenados en el buffer, siempre y cuando sean de un tamaño mayor que 250 Bytes, ya que estos serán descartados, y enviados directamente hacia IP sin pasar por la capa *network-coding*.

- Codificar: en esta función se llevará a cabo la parte nuclear del trabajo, que es la codificación de los paquetes que se van almacenando en el buffer de transmisión. Como ya se ha mencionado anteriormente, esta función es llamada por ReceiveFromUpperLayer, una vez que se ha alcanzado un determinado número K de paquetes acumulados en el buffer. Es en esta función donde se creará la cabecera de *network-coding* mediante MoreHeader añadiéndola al paquete. Una vez hecho esto, este será enviado hacia el nivel IP.
- EnvíoSinCodificar: a esta función se puede llegar por dos puntos diferentes. Si se supone que ya se ha recibido la totalidad de los paquetes desde aplicación, habiéndose producido la codificación de los mismos, y aún quedan paquetes en el buffer de transmisión sin codificar; siempre y cuando estos sean menos que K, se realizará la llamada a esta función y a la totalidad de dichos paquetes se les añadirá también su cabecera de *network-coding* y se procederá a enviarlos hacia el nivel inferior, pero codificados. La otra posibilidad que hay para llegar a esta función es en el supuesto de que se estén recibiendo paquetes en ReceiveFromUpperLayer y, a su vez, se almacenen en el buffer de transmisión, pero no se llega al número mínimo de K paquetes almacenados, por lo que pasado un determinado tiempo, ante la falta de paquetes y por tanto la imposibilidad de codificar, se procederá también a realizar la llamada a esta función para nuevamente añadirles la cabecera *network-coding* y enviarlos hacia abajo.
- GenerateRandomVector: como su propio nombre indica, esta función creará un vector aleatorio de K valores de 0 y 1, ya que se trabaja con GF(2). Estos vectores aleatorios son los que se irán añadiendo a la cabecera de *network-coding* y, a su vez, serán utilizados para realizar la codificación y decodificación de los paquetes.
- Receive: se encargará de ir recibiendo y almacenando los paquetes que van llegando al receptor, para más tarde proceder a su decodificación llamando a la función Decodificar, en el caso de que sean paquetes codificados, o directamenteme enviarlos hacia el nivel UDP, en el caso de que sean paquetes que llegan sin codificar. Es en esta función donde se usan los vectores aleatorios creados anteriormente para ir almacenándolos en una matriz que se utilizará en la función Decodificar. Otra de sus funciones es, en el nodo transmisor, procesar los ACK de confirmación enviados por el receptor.

- Decodificar: esta función tiene varios objetivos; en primer lugar se producirá la decodificación mediante el cálculo de la matriz inversa de vectores aleatorios que se han ido creando en el Receive; una vez hecho esto, se mandarán los paquetes almacenados en el buffer de recepción hacia el nivel superior. En segundo lugar, se mandará el ACK hacia el transmisor para informar de que la decodificación se ha realizado de manera exitosa, y no hace falta continuar con el envío de paquetes.

4

Desarrolo en el marco de NS-3

Este capítulo se sigue centrando en la herramienta utilizada para la realización de este trabajo, es decir, en NS-3. Se describirá y explicarán los principales módulos que se han desarrollado.

4.1 Módulos empleados en NS-3

A lo largo de este capítulo se describen todos los módulos que componen la implementación de la técnica de *network-coding* para NS-3. Se han situado en la carpeta /src/network-coding, y han sido programados en C++, por lo que sería bastante costoso entrar a explicarlos en detalle a nivel de código, así que se hará una descripción de cada uno de ellos.

4.1.1. Cabecera Network Coding

En este apartado se explicará uno de los aspectos principales, como es la cabecera usada por este nuevo protocolo. La creación de la cabecera se lleva a cabo en el fichero more-header, el cual se encuentra en /src/network-coding/model/more-header(.cc/.h). La cabecera *network-coding* se añade a un paquete, cada vez que haya en el buffer de transmisión, como mínimo K paquetes acumulados. Una vez se llegue o se supere ese número de paquetes recibidos desde la aplicación, se producirá una llamada a la función Codificar que, además de realizar la codificación, irá añadiendo a cada paquete su correspondiente cabecera. Evidentemente el nodo encargado de codificar y añadir la cabecera será el nodo transmisor.

Tras una serie de cambios y diversas pruebas para garantizar la máxima eficiencia, se llegó a la conclusión de que la cabecera estaría formada únicamente por seis campos, de los cuales cinco serán fijos y uno variable (Figura 4.1):

1. Valor de K: este primer campo es de longitud fija y ocupa 1 Byte; sin duda es uno de los elementos principales de la cabecera, ya que indica el número de paquetes que se han codificado entre sí para dar lugar al paquete único que será enviado al nodo receptor; obviamente será una combinacion lineal de los K paquetes extraídos del buffer de transmisión. Cabe destacar que este valor además es necesario para que el nodo receptor pueda llevar a cabo la decodificación de los paquetes que llegan, y así obtener, mediante la matriz inversa (será de dimensión K x K), los paquetes originales. Hay que destacar una circunstancia especial con respecto a este campo, y es que en algunos casos llevará el valor K=0; esto se dará cuando se esté transmitiendo paquetes que han pasado por la capa de *network-coding* pero que no han sido codificados, y viajan como paquetes originales, debido a que el buffer no llega al valor mínimo de K paquetes para codificar.

2. Número de fragmento: este campo se refiere al número del fragmento que se está enviando, su longitud también es fija de 2 Bytes. Su valor será esencial para que los nodos transmisor y receptor mantengan la sincronía respecto al fragmento, lo que se producirá cuando el nodo receptor haya completado su matriz de K x K y pueda realizar su inversa para recuperar los paquetes originales. Una vez hecho esto, el nodo receptor mandará un paquete de reconocimiento al nodo transmisor, avisándole de que ya ha recuperado los primeros K paquetes y que, por tanto, necesita que comience a enviar los siguientes K paquetes codificados, es decir, el siguiente fragmento, lo que se indicará en la cabecera. Si se supone que desde aplicación llegan 20 paquetes, se guardan en el buffer de transmisión, para K=10, entonces se tendrán 2 fragmentos: el fragmento 0 y el fragmento 1.

3. Paquete codificado o paquete de aviso: este campo es también de longitud fija de 1 Byte y su funcionalidad es la de informar a los nodos mediante 0 ó 1 si lo recibido es un paquete normal codificado (0) o, por el contrario, un paquete de confirmación o ACK (1) para el nodo transmisor. En el caso de que llegue un 0 el nodo receptor lo considerará un paquete codificado y procederá a analizarlo, mientras que si fuera un 1, entonces el nodo transmisor comenzaría a enviar el siguiente fragmento.

4. Puerto origen y destino: para indicar los puertos de origen y destino se usa también un campo fijo de 2 Bytes para cada uno.

5. Vector aleatorio: este es el útimo campo de la cabecera que introduce la capa de *network-coding*, y su longitud es variable desde un tamaño mínimo de 1 Byte hasta un máximo de $\frac{K}{8}$ Bytes, dependiendo del valor de K. Como su propio nombre indica, en este campo se sitúa el vector aleatorio de unos y ceros generado y utilizado para codificar los respectivos paquetes originales. Estos vectores irán almacenándose en recepción, y se irá completando una matriz de K x K siempre y cuando el vector que llega sea linealmente independiente de los que ya hay, ya que si no lo es, se deshecha el paquete puesto que no proporciona nueva información.

Es importante aclarar un aspecto: es que cuando se habla de un paquete que llega al nodo receptor sin codificar, llevará $K = 0$ y el campo de $paquetecodificado = 0$.

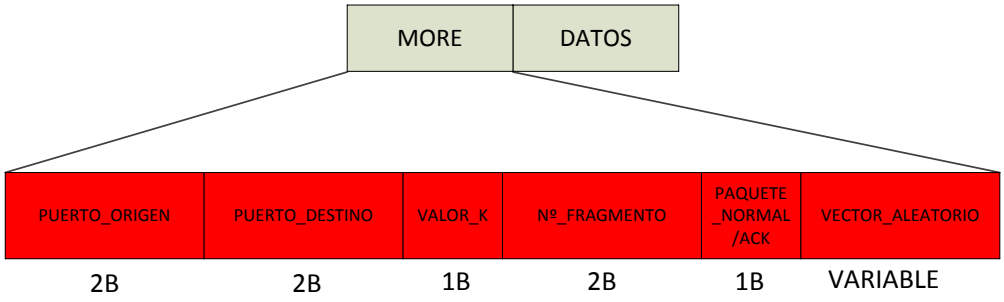


Figura 4.1: Cabecera *network-coding*

Respecto a la parte variable de la cabecera de *network-coding*, se puede comprobar en la Figura 4.2 que el tamaño dependerá del valor de K. Este campo ocupará $\frac{K}{8}$ Bytes, como ya se ha mencionado anteriormente, y en el caso de que falten bits para completar el último Byte, se rellenará con ceros.

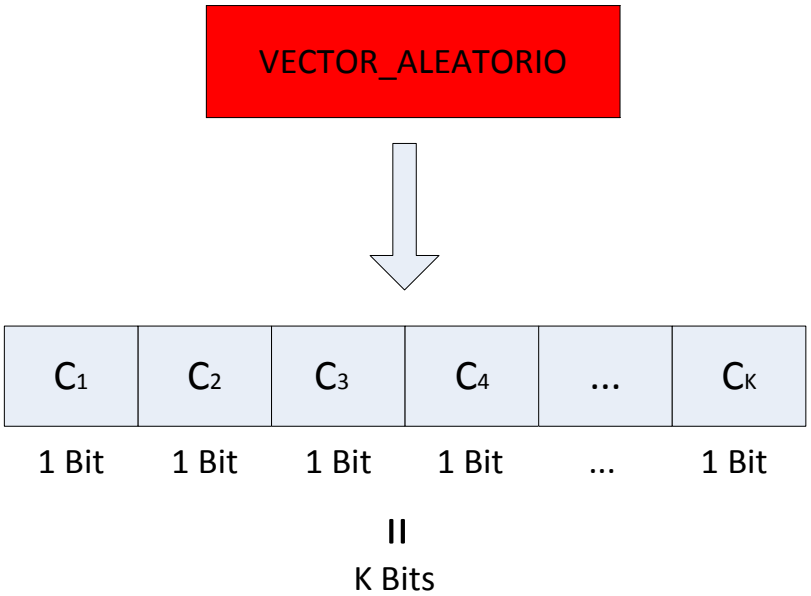


Figura 4.2: Parte variable cabecera

4.1.2. Proceso de codificación

Al hablar de *network-coding* hay que tener una idea muy clara, y es que como ya se ha dicho en numerosas ocasiones, los paquetes no viajarán sin más, sino que empezarán a sufrir cambios, en este caso, al ser codificados unos con otros, proceso que, en este trabajo llevará a cabo el propio nodo transmisor. Los pasos que se siguen para realizar la codificación son los siguientes:

1. Se supone que la capa de aplicación comienza a enviar paquetes hacia abajo; al llegar a la capa de *network-coding*, se empiezan a acumular en el buffer siempre y cuando sean mayores que un determinado tamaño (200 Bytes), ya que si son menores siguen su proceso habitual.

2. Una vez se han acumulado en el buffer de transmisión un determinado número de paquetes=K, se dispara un evento para comenzar el proceso de codificación. El hecho de que el buffer haya alcanzado ese número mínimo de K paquetes para poder comenzar la codificación no implica que no puedan seguir llegando paquetes, y se continúen acumulando en el buffer.

3. A continuación se muestra cómo se opera con ellos; se supone que los paquetes P_1,P_2,P_3...P_k son los nativos que van llegando al buffer, a partir de los cuales se crearán los nuevos paquetes codificados P_1^*,P_2^*,P_3^*...P_k^*.

$$P_n^* = \sum_{i=1}^{K} C_{n,i} P_i$$

$$\begin{aligned}
P_1^* &= C_{1,1}P_1 + C_{1,2}P_2 + C_{1,3}P_3 + ... + C_{1,K}P_K \\
P_2^* &= C_{2,1}P_1 + C_{2,2}P_2 + C_{2,3}P_3 + ... + C_{2,K}P_K \\
P_3^* &= C_{3,1}P_1 + C_{3,2}P_2 + C_{3,3}P_3 + ... + C_{3,K}P_K \\
&. \\
&. \\
&. \\
P_k^* &= C_{K,1}P_1 + C_{K,2}P_2 + C_{K,3}P_3 + .. + C_{K,K}P_K
\end{aligned}$$

Donde los C_i son los coeficientes del vector aleatorio generado (0 ó 1)

Es importante destacar que en el sumatorio que se muestra previamente, he utilizado P_K^*, debido a que realmente el número mínimo de envíos será K, pero esto sucederá en el mejor de los casos (si no hay errores), y todos los vectores aleatorios empleados son linealmente independientes de los anteriores; en caso contrario serán necesarios un número de envíos mayor que K.

4.1.3. Proceso de decodificación

Una vez llevado a cabo el proceso de codificación explicado en el punto anterior, los paquetes se envían y, a través del medio inalámbrico, llegan al nodo receptor, el cual será el encargado de llevar a cabo la decodificación de los mismos, para finalmente recuperar los paquetes originales.

A lo largo de este apartado se explica el proceso más detalladamente de cómo funciona el mecanismo de decodificación. Como ya se ha dicho, la capa de *network-coding* se sitúa por encima de IP, por lo que los niveles inferiores trabajan con normalidad.

Una vez se comienzan a recibir paquetes en el nodo receptor, se siguen una serie de pasos que se detallan a continuación:

1. En primer lugar se comprueba el tipo de paquete que ha llegado, esto es, si se trata de un paquete sin codificar (con un 0 en el campo de paquete de aviso o paquete codificado) y, por lo tanto, deberá ser mandado hacia el nivel superior tal cual o, al contrario, si se trata de un paquete codificado.

2. Bajo el segundo supuesto, en el que llega un paquete codificado, (se reconoce por la cabececera *network-coding*, dado que en el campo de paquete codificado o paquete de aviso llevará un 0, y un valor distinto de 0 en el campo "valor de K"). Se procederá a almacenar o no el paquete en el buffer de recepción, en función de unas premisas:

   - Si una vez recuperado el vector aleatorio de la cabecera se comprueba que se trata de un vector linealmente independiente de los que previamente se habían guardado, entonces se añadirá el vector a la matriz de decodificación, y se almacenará el paquete en el buffer, considerandolo útil.
   - Si es linealmente dependiente de los demás, entones se procederá a su descarte, ya que no aporta información nueva para poder realizar la matriz inversa, y así obtener los paquetes originales.

3. Una vez se completa la matriz de K x K, con los K vectores aleatorios linealmente independientes, se lleva a cabo la decodificación. De esto se encarga una función específica que, a su vez, limpia del buffer de recepción los paquetes ya decodificados. En este proceso de cálculo se generan, por una parte, los paquete originales, y por otra, el vector aleatorio junto con los paquetes codificados almacenados en el buffer de recepción. A continuación se observa la expresión que permite obtener los paquetes nativos u originales mediante la matriz inversa:

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ . \\ . \\ . \\ P_K \end{bmatrix} = \begin{bmatrix} C_{1,1} & C_{1,2} & C_{1,3} & ... & C_{1,K} \\ C_{2,1} & C_{2,2} & C_{2,3} & ... & C_{2,K} \\ C_{3,1} & C_{3,2} & C_{3,3} & ... & C_{3,K} \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ ... & ... & ... & ... & ... \\ C_{K,1} & C_{K,2} & C_{K,3} & ... & C_{K,K} \end{bmatrix}^{-1} \begin{bmatrix} P_1^* \\ P_2^* \\ P_3^* \\ . \\ . \\ . \\ P_K^* \end{bmatrix}$$

Una vez descritos los procesos de codificación y decodificación, se representa un ejemplo ilustrativo con $K = 4$ en la Figura 4.3. En este caso, se puede ver, en función de la escala de tiempos, el proceso de envío de los paquetes hasta su correcta recepción por parte de la aplicación destino.

En este ejemplo se ha supuesto un K bajo, pero lo esencial y en lo que mayor hincapié se ha de hacer es que se cuenta con tres gráficas distintas, cada una de ellas representa distintos momentos temporales y distintos estados de los paquetes en las capas del modelo. Como se puede apreciar en la gráfica del receptor, una vez se han recibido los 4 paquetes codificados y linealmente independientes (se supone un caso ideal), se envía, en ese preciso instante, hacia el nodo transmisor un ACK, para confirmar la correcta recepción del fragmento.

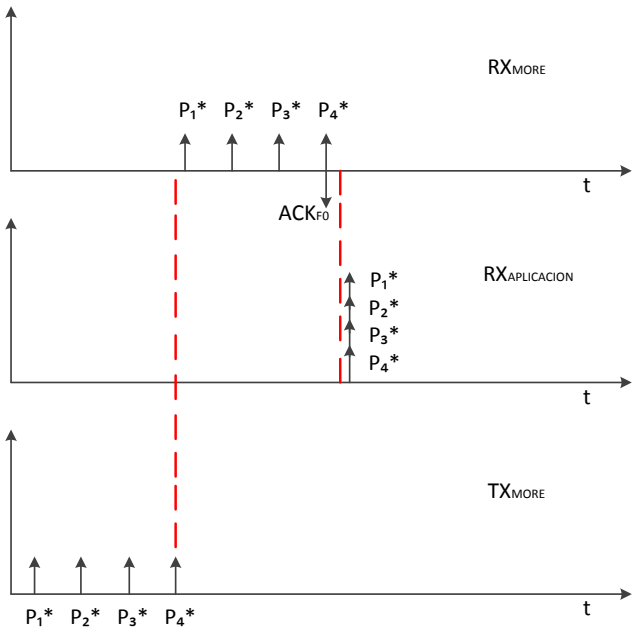


Figura 4.3: Proceso de transmisión

4.2 Archivos de configuración del canal

En este apartado se destaca el papel relevante que juegan unos determinados archivos, la clase ConfigureScenario, la cual se encargará fijar los parámetros iniciales de la simulación para que esta se complete de manera satisfactoria. Esto se hará a través de cuatro ficheros: *network-coding-scenario.conf*, *two-nodes-scenario.conf*, *two-nodes-channel.conf* y *two-nodes-static-routing.conf*:

- El fichero *network-coding-scenario.conf* es un archivo de texto plano que se utiliza de manera permanente, ya que se encarga de establecer los principales parámetros de la configuración del escenario y de la simulación (véase la Figura 4.4) , como son el número de simulaciones, el número de paquetes transmitidos, la longitud de los datos, la FER, el protocolo de transporte, el tipo de escenario, la presencia de *network-coding*, el valor de K, los archivos traza de salida, etc.

```
[SCENARIO]
RUN=50
RUN_OFFSET=0
NUM_PACKETS=100
PACKET_LENGTH=1472
FER=0.0

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

```
[NETWORK_CODING]
ENABLED=1
PROTOCOL=MoreNetworkCodingProtocol
BUFFER_SIZE=10
BUFFER_TIMEOUT=100
MAX_CODED_PACKETS=2
EMBEDDED_ACKS=0
GF=1
K=64

[OUTPUT]
APPLICATION_TRACING=0
TCP_TRACING=0
TCP_SHORT_TRACING=0
NETWORK_CODING_TRACING=1
NETWORK_CODING_SHORT_TRACING=1
```

Figura 4.4: Fichero de configuración

- Ell archivo *two-nodes-scenario.conf* define la posición física de los nodos, incluyendo su identificador, coordenadas (x,y,z), si se trata del nodo transmisor (TX), receptor (RX) o, nodos encargados de reenviar paquetes (FWD). Como los escenarios analizados cuentan únicamente con 2 nodos, la configuración es muy sencilla, como se ve en la Figura 4.5.

```
#No.    X       Y       Z       TX      RX      CR      FWD
1       0       0       0        2       0       0       1
2       5       0       0        0       1       0       1
```

Figura 4.5: Fichero localización física de los nodos

- El archivo *two-nodes-channel.conf* sirve para realizar la configuración necesaria del canal como se observa en la Figura 4.6, y sus valores varían de la siguiente manera:

- 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 archivo de configuración inicial: *network-coding-scenario.conf*.
- 1 - No hay enlace. Es un enlace caído.

```
1       5
0       1
```

Figura 4.6: Fichero de configuración del canal

- Por último el fichero *two-nodes-static-routing.conf* (véase la Figura 4.7) nos servirá para realizar la configuración de encaminamiento de los nodos. En este caso es necesario fijar para cada nodo el destino, el próximo salto e interfaz.

```
#Node ID        Destination     Nexthop         Interface
0               1               1               1
1               0               0               1
```

Figura 4.7: Fichero de configuración de encaminamiento

4.3 Ficheros traza de salida

Una vez se han realizado las simulaciones, es necesario ver los resultados obtenidos para poder procesarlos, y esto se consigue gracias a los ficheros de traza que se van rellenando a medida que se ejecutan las simulaciones de manera automática. Obtener trazas para ser procesadas es una de las necesidades que se presentan en el proyecto.

Aunque existen diversas formas de obtener los datos que son objeto de estudio, se ha optado por desarrollar una nueva clase, *ProprietaryTracing*. Ésta recoge una serie de mecanismos para poder representar en ficheros los parámetros deseados. Para poder tener un mayor control sobre los resultados a obtener se puede seleccionar entre varios tipos de trazas. Éstos se filtrarán posteriormente para, mediante Matlab, representar las gráficas correspondientes.

El primer tipo de traza se la denomina "larga"(véase la Figura 4.8), ya que para cada simulación refleja, con sus respectivos datos, cada uno de los paquetes transmitidos. Esta traza permite comprobar el correcto funcionamiento del sistema implementado, y los campos que tiene son los siguientes:

- Tiempo: como es obvio proporciona el tiempo que va transcurriendo durante la simulación, partiendo siempre de 20 segundos. Este valor es necesario para el cálculo del throughput.

- Estado: este campo se utiliza para aclarar en qué función del código se encuentra la simulación en cada momento; esto es, para saber si el paquete se encuentra en fase de codificación, si se acaba de recibir en recepción, etc. Existen seis posibilidades:
  - Estado=0. Proceso de codificación en el nodo transmisor.
  - Estado=1. Esto significará que es un paquete que se envía sin codificar (EnvíoSinCodificar).
  - Estado=2. En este caso indica que un paquete codificado acaba de llegar al Receive del nodo receptor.
  - Estado=3. Este estado establece que el nodo transmisor acaba de recibir un paquete de aviso o ACK.
  - Estado=4. Cuando el paquete se encuentra en fase de decodificación.
  - Estado=5. Este último estado indica que un paquete enviado desde EnvíoSinCodificar llega al Receive del nodo receptor, que puede así distinguirlo de un paquete codificado (estado=2).
- NodoId: este campo, como su nombre indica, informa de la identidad del nodo, si se trata del nodo transmisor (0) o receptor (1).
- IpOrigen/IpDestino: identifica las direcciones ip origen, y destino que se mantienen constantes.
- TamañoPaquete: en este apartado se informa a cerca del tamaño que tiene el paquete transmitido.
- Nº Fragmento: como ya se mencionó en el capítulo anterior, el número de fragmento permite identificar aquel al que pertenece el paquete codificado.

A continuación, se pueden observar esta serie de parámetros, en un claro ejemplo de "traza larga"que se ha conseguido mediante la ejecución de una simulación. Como ya se ha mencionado, las direcciones ip de ambos nodos, siempre van a ser las mismas; mientras que, en cuanto al tamaño de los paquetes, se puede ver que son de 1000 Bytes a excepción únicamente de los ACK que son de 4 Bytes.

Tiempo	Estado	NodoId	IpOrigen	IpDestino	TamañoPaquete	N°Fragmento
20.000320	0	0	10.0.0.1	10.0.0.2	1000	0
20.001292	2	1	10.0.0.1	10.0.0.2	1000	0
20.005320	0	0	10.0.0.1	10.0.0.2	1000	0
20.006292	2	1	10.0.0.1	10.0.0.2	1000	0
20.010320	0	0	10.0.0.1	10.0.0.2	1000	0
20.011292	2	1	10.0.0.1	10.0.0.2	1000	0
20.015320	0	0	10.0.0.1	10.0.0.2	1000	0
20.016292	2	1	10.0.0.1	10.0.0.2	1000	0
20.020320	0	0	10.0.0.1	10.0.0.2	1000	0
20.021292	2	1	10.0.0.1	10.0.0.2	1000	0
20.025320	0	0	10.0.0.1	10.0.0.2	1000	0
20.026292	2	1	10.0.0.1	10.0.0.2	1000	0
20.026292	4	1	10.0.0.1	10.0.0.2	1000	0
20.026292	4	1	10.0.0.1	10.0.0.2	1000	0
20.026292	4	1	10.0.0.1	10.0.0.2	1000	0
20.026292	4	1	10.0.0.1	10.0.0.2	1000	0
20.026892	3	0	10.0.0.2	10.0.0.1	4	1
20.030320	0	0	10.0.0.1	10.0.0.2	1000	1
20.031292	2	1	10.0.0.1	10.0.0.2	1000	1
20.035320	0	0	10.0.0.1	10.0.0.2	1000	1

Figura 4.8: Formato de fichero

En esta captura se observa el instante en el que comienza la transmisión de paquetes y el momento en el que el nodo receptor una vez ha completado la matriz de K x K, procede a la decodificación de los paquetes (Estado=4,NodoId=1) para obtener la información original. El nodo transmisor recibe el paquete de aviso o ACK (Estado=3,NodoId=0), momento en el que el N° Fragmento se incrementa en uno, y nuevamente se comienza a enviar el siguiente grupo de paquetes codificados.

Este tipo de trazas resultó muy útil durante el proceso inicial del trabajo, para comprobar el correcto funcionamiento del desarrollo, pero una vez verificado este punto, se desarrolló otro tipo de trazas que no ocuparan tanto espacio para cada simulación, con un resultado más global de cada experimento. De esta manera surgió lo que se ha venido a llamar trazas resumidas, que como se verá a continuación dan información bastante más compleja acerca de cada simulación:

- No: indica el numero de simulación.
- K: informa del valor de K en la simulación.
- Throughput: valor del throughput, calculado como el cociente de los datos útiles entre el tiempo de transmisión empleado para transmitirlos.
- Transmissions: número de transmisiones que se realizan, esto es, el número total de paquetes que llegan a recepción.

- TiempoTotal: el tiempo empleado para llevar a cabo la tranmisión de toda la información.
- BytesTotales: cantidad de datos útiles a nivel de aplicación.
- MedTiempos: calcula, para cada simulación, la media del retardo entre paquetes consecutivos.
- VarTiempos: calcula la varianza del retardo entre paquetes consecutivos.
- MedInv: calcula la media de los tiempos empleados al realizar el cálculo de la matriz inversa.
- VarInv: calcula la varianza de los tiempos empleados para obtener la matriz inversa.
- MedRango: nos da la media de los tiempos usados para calcular el rango de la matriz, y comprobar así si el vector aleatorio añadido es linealmente independiente de los demás o no.
- VarRango: la varianza de los tiempos empleados para calcular el rango de la matriz.

No.	K	FER	Throughput	Transmissions	TiempoTotal	BytesTotales
1	2	0.00	3115.33643	16809	37.620335	14650000
2	2	0.00	3126.96753	16753	37.480404	14650000
3	2	0.00	3165.08740	16515	37.028996	14650000

MedTiempos	VarTiempos	MedInv	VarInv	MedRango	VarRango
0.0075256	0.0000108	0.0083314	0.0000262	0.0149393	0.0046347
0.0074976	0.0000101	0.0086366	0.0000671	0.0144840	0.0040435
0.0074073	0.0000098	0.0088226	0.0000720	0.0148884	0.0042580

Figura 4.9: Formato de fichero

4.4 Automatización de la simulación

Para llevar a cabo la automatización de las simulaciones se ha creado un archivo de texto (Figura 4.10) que se usa a modo de script; no es más que un simple programa que automatiza la ejecución de un número elevado de simulaciones donde a su vez mediante diversos bucles se irán variando determinados parámetros de la simulación. En este trabajo se han realizado 100 experimentos por cada caso, lo que da lugar a 7200 simulaciones.

Dicho script está basado en un bucle continuo en el que se generan todas las combinaciones necesarias para los escenarios a analizar. Se ejecutará el archivo *test-scenario*, cuyos parámetros de entrada se localizan en el fichero mencionado anteriormente, *network-coding-scenario.conf*, los cuales van a verse modificados por las variables de los bucles. Finalmente este script irá generando, en otro fichero, las trazas resumidas.

```
#! /bin/bash

  for K in 2 4 8 16 32 64 128 255
   do
    for fer in 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
     do
      for runoffset in 0 20 40 60 80
       do
        ./waf --run "scratch/test-scenario --Fer=$fer --K=$K --Runoffset=$runoffset"
       done
     done
   done
```

Figura 4.10: Script para simulaciones

5

Simulaciones y resultados

En este último capítulo se van a establecer, en primer lugar, una serie de conceptos que se consideran claves y necesarios para poder comprender los resultados que se presentarán posteriormente. A continuación se explican los parámetros mediante los cuales se han realizado las simulaciones para pasar a presentar los resultados obtenidos en el escenario estudiado, tanto con tráfico UDP como con TCP. Para finalizar se destacarán las mejoras obtenidas al utilizar *network-coding*, y UDP junto con la nueva técnica utilizada (MORE).

5.1 Conceptos fundamentales

Los conceptos claves que son necesarios para interpretar adecuadamente los resultados son los que se enumeran seguidamente.

- Throughput: se define como el cociente entre los datos útiles correctamente recibidos en el equipo destino y la duración de la transmisión. Es uno de los principales parámetros de estudio, puesto que aporta información básica acerca del rendimiento de la red.

$$Throughput = \frac{\text{Total bits de datos}}{\text{Tiempo de simulación}}$$

- Retardo: parámetro que indica el retardo medio entre paquetes. Para su cálculo se registran todos los instantes temporales de recepción correcta de un paquete para calcular su valor medio.
- Número de transmisiones: variable que indica el número de transmisiones que han sido necesarias para completar un envío. Es importante para estimar el ahorro energético que supone emplear *network-coding*.
- K: parámetro que indica el número de paquetes a codificar en cada caso.

- Tinversa: tiempo empleado por el simulador para realizar la operación de la matriz inversa.
- Trango: tiempo necesario para realizar el cálculo del rango de la matriz.

5.2 Proceso de medida

Este apartado detalla las decisiones que se han tomado para obtener los resultados de las simulaciones que se describen en las siguientes páginas. En algunos casos es evidente que la cantidad de información que se genera durante las simulaciones hace inviable obtener medidas detalladas, siendo imposible además revisar los resultados.

- Se han realizado un total de 7200 ejecuciones independientes sobre el escenario simulado, considerando en cada una de ellas que se generan, desde la aplicación, 10000 paquetes de datos. Como es lógico se han realizado simulaciones para diversos valores de FER, desde $FER = 0.0$ hasta $FER = 0.8$ con un paso de 0.1 y, a su vez, para $K = 2, 4, 8, 16, 32, 64, 128, 255$, lo que da el total de 7200. En las gráficas posteriores se muestra el valor medio, acompañado de un intervalo de confianza del 95 %, obtenido mediante la siguiente expresión:

$$Intervalo\ de\ confianza = 2.2281 \cdot \frac{\sqrt{\text{Varianza de las medidas}}}{\sqrt{\text{Número de medidas}}}$$

- Todos los paquetes tendrán el tamaño máximo permitido por la conexión, que está fijado en 1500 Bytes. Se tendrá en cuenta que además del tamaño de las cabeceras de transporte y red ($IP = 20Bytes$), también se ha de considerar la cabecera introducida por la capa de *network-coding*:
  - Si se utiliza UDP (8 Bytes) como protocolo de transporte, el tamaño máximo de los datos será de $1500 - 20 - 8 = 1472$ y, teniendo en cuenta la cabecera de *network-coding*, se obtienen $1472 - \left(8 + \frac{K}{8}\right) Bytes$
  - En el caso de utilizar como protocolo de transporte TCP (20 Bytes), el tamaño máximo de los datos será de $1500 - 20 - 20 = 1460Bytes$.
- Todas las transmisiones se hacen a un régimen binario de de 11 Mbps, mientras que la tasa básica es de 1 Mbps, emulando la tecnología IEEE 802.11b.
- Para simplificar el estudio se ha decidido establecer que los ACK a nivel MAC 802.11 nunca se pierden, y que se transmiten a la velocidad básica, es decir, 1 Mbps.
- El canal está saturado, asegurando que el transmisor siempre tiene paquetes esperando para ser enviados.

- Para todos los escenarios se ha utilizado, como ya se ha adelantado, enlaces inalámbricos bajo la tecnología IEEE 802.11b.

5.3 Escenario estudiado

En este apartado se describe el escenario que se ha utilizado en el trabajo para realizar los experimentos. Se detallarán sus principales características, así como los parámetros fundamentales que son necesarios conocer para poder comprender correctamente los resultados que más adelante serán analizados.

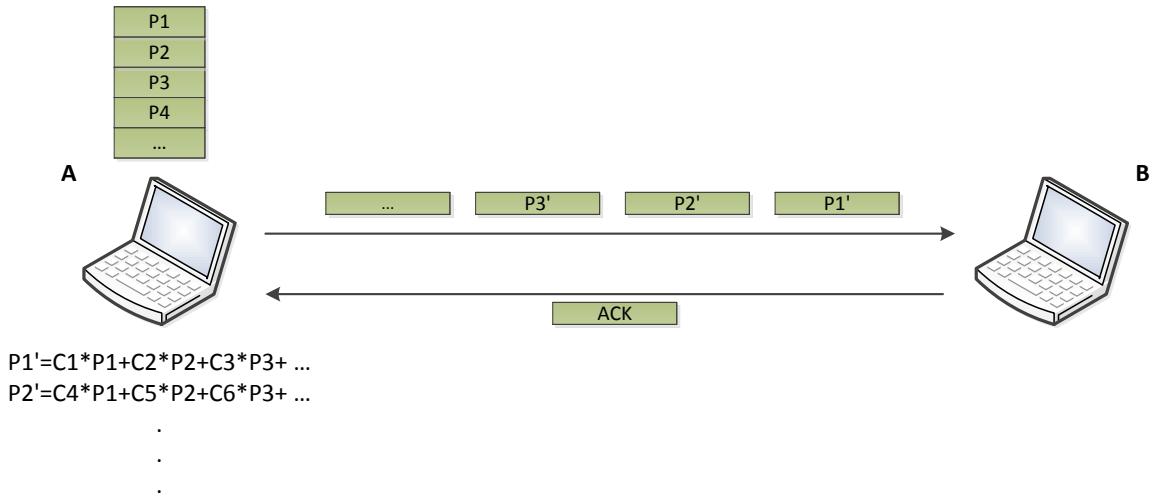


Figura 5.1: Escenario simulado

Se va a realizar la totalidad de las pruebas sobre un único escenario que estará formado por dos nodos: transmisor y receptor, tal y como recoge la Figura 5.1, en la que los equipos, el equipo A y el B, harán indistintamente de transmisor o receptor. Se van a considerar los dos únicos flujos posibles: A-B y B-A; en una dirección se realizará el envío de datos y en la otra el de los reconocimientos.

A continuación se describe paso a paso el funcionamiento de la implementación de *network-coding* empleada:

1. En primer lugar se comienza a almacenar los paquetes de datos (P1,P2,P3...) en el buffer de transmisión del equipo A, hasta el momento en el que se llega a un determinado número de paquetes igual a K.

2. Una vez alcanzado dicho valor, se comienza a realizarse la codificación de los K paquetes previamente almacenados en el buffer de transmisión. Como se puede ver en la Figura 5.1, la codificación se lleva a cabo en el nodo transmisor, y los respectivos paquetes no serán eliminados del buffer hasta el momento en que se certifique la correcta recepción y decodificación de los mismos.

3. Una vez iniciado el proceso de codificación, se comenzará a realizar el envío continuo de paquetes codificados (P1´,P2´,P3´...), que, a su vez, se irán almacenando en el buffer de recepción, a la vez que se va formando la matriz con los que son linealmente independientes, ya que los demás serán eliminados al no aportar información nueva.

4. En el momento en el que se complete la matriz de K x K , se habrán recibido correctamente K paquetes codificados, y se procederá a enviar un paquete hacia el nodo transmisor a modo de ACK, para confirmar la correcta recepción de la información. Posteriormente se procederá a eliminar del buffer de transmisión los K paquetes iniciales, para continuar con los K siguientes. El nodo receptor se encargará de realizar el proceso de decodificación con ayuda de la matriz inversa y los paquetes codificados almacenados en el buffer, de manera que recuperará los originales.

Un factor muy importante a tener en cuenta en el desarrollo del proyecto ha sido la ayuda que ha supuesto utilizar la librería IT++, la cual ha sido de gran utilidad para los cálculos con matrices y vectores sobre el cuerpo finito GF(2), en la que $0 + 0 = 1 + 1 = 0$ y $0 + 1 = 1 + 0 = 1$, es decir, igual a una operación lógica XOR. IT++ es una librería en C++ de clases y funciones matemáticas , procesado de señales y estadísticas muy utilizada. Su uso principal es la simulación de sistemas de comunicación para la realización de investigaciones en el área de las comunicaciones. El núcleo de la biblioteca consta de un tipo de vector genérico, clases de la matriz, y un conjunto de rutinas que lo acompañan; dicho núcleo hace que IT++ se asemeje a MATLAB , GNU Octave o SciPy.

El origen de esta biblioteca se encuentra en el antiguo departamento de Teoría de la Información en la Universidad Tecnológica de Chalmers y está siendo desarrollada, y ampliamente utilizada, por los investigadores que trabajan en el área de las comunicaciones. En 2005 , 2006 y 2007 , IT++ fue desarrollada como parte de la Red Europea de Excelencia en Comunicaciones Inalámbricas (NEWCOM).

5.3.1. Resultados de las simulaciones

En este apartado se explican los resultados más relevantes de las simulaciones realizadas. A lo largo del desarrollo de este trabajo se han hecho multitud de medidas, se van a mostrar los parámetros que se consideran más significativos, de forma que pueda apreciarse la influencia de la técnica de codificación implementada. Se realizará un estudio y análisis gráfico de los resultados, para finalmente llegar a una serie de conclusiones.

Throughput en canal ideal

En primer lugar se estudiará el que posiblemente sea el principal parámetro de interés, el throughput. En la Figura 5.2 se puede ver el throughput en función de los distintos valores de K empleados en el trabajo. En este primer resultado se puede comprobar que la configuración que da un mejor rendimiento es $K = 64$ (en torno a unos 5500 Kbps); aumentar el valor de K es beneficioso para limitar las transmisiones innecesarias, puesto que a mayor valor de K se tendrán menos posibilidades de obtener vectores linealmente dependientes reduciendo, por tanto, el tiempo empleado en la transmisión de información. Por otra parte, se debe destacar que el inconveniente de incrementar K implica tener menor capacidad para el envío de datos útiles, ya que la cabecera de *network-coding* será mayor conforme vaya aumentando dicho valor. Es por esto que a partir de $K = 64$ no se mejora el rendimiento.

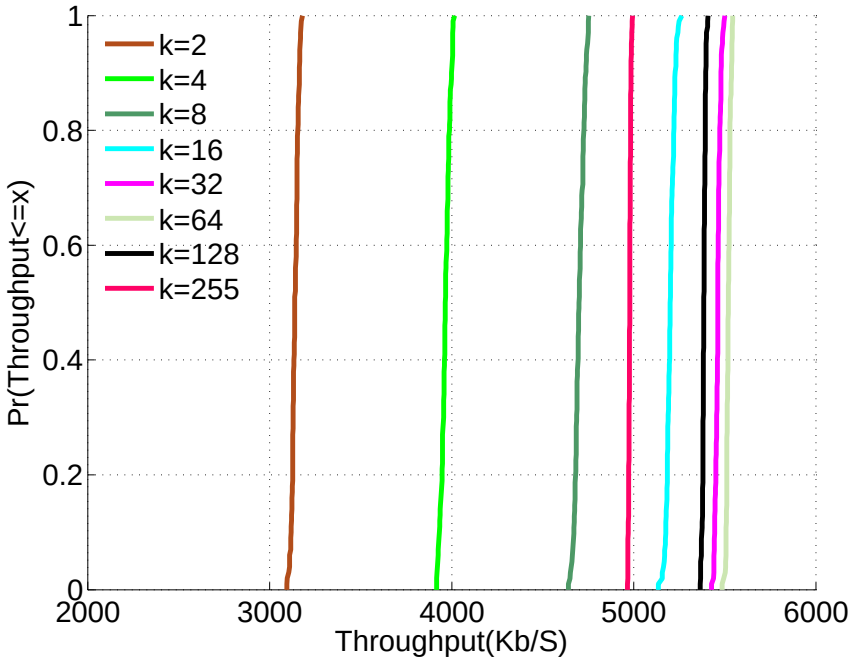


Figura 5.2: Función distribución del throughput para la cabecera inicial

Así, la influencia que tiene la sobrecarga repercute de manera directa sobre el throughput. En un prinicipio se estableció una cabecera de *network-coding* con un tamaño del campo del vector aleatorio de K Bytes, es decir, ocupando un Byte para cada valor de 0 ó 1, obteniéndose los resultados que muestra la Figura 5.2. A la vista de los mismos, se estudió la posibilidad de mejorar este rendimiento realizando una modificación en dicha cabecera, haciendo que el campo del vector aleatorio fuese variable, de manera que:

- Para $K = 2$ hay un espacio libre para datos de $1472 - \left(8 + \frac{2}{8}\right) = 1463 Bytes$, mientras que antes era de $1472 - (8 + 2) = 1462 Bytes$. Se obtiene una mejora de 1 Byte.

- Para $K = 4$ el espacio libre para datos es de $1472 - \left(8 + \frac{4}{8}\right) = 1463 Bytes$, frente a los $1472 - (8 + 4) = 1460 Bytes$ de antes, obteniendo una mejora de 3 Bytes.
- Para $K = 8$ queda un espacio libre para datos de $1472 - \left(8 + \frac{8}{8}\right) = 1463 Bytes$, mientras que antes era de $1472 - (8 + 8) = 1456 Bytes$. Se obtiene una mejora de 7 Bytes.
- Para $K = 16$ hay un espacio libre para datos de $1472 - \left(8 + \frac{16}{8}\right) = 1462 Bytes$, mientras que antes era de $1472 - (8 + 16) = 1448 Bytes$. Se obtiene una mejora de 14 Bytes.
- Para $K = 32$ queda un espacio libre para datos de $1472 - \left(8 + \frac{32}{8}\right) = 1460 Bytes$, mientras que antes era de $1472 - (8 + 32) = 1432 Bytes$. Se obtiene una mejora de 28 Bytes.
- Para $K = 64$ el espacio libre para datos es de $1472 - \left(8 + \frac{64}{8}\right) = 1456 Bytes$, frente a los $1472 - (8 + 64) = 1400 Bytes$ de antes, obteniendo una mejora de 56 Bytes.
- Para $K = 128$ la capacidad libre para datos es de $1472 - \left(8 + \frac{128}{8}\right) = 1448 Bytes$, y antes era de $1472 - (8 + 128) = 1336 Bytes$, por lo que se obtiene una mejora de 112 Bytes.
- Para $K = 255$ se tiene un espacio libre para datos de $1472 - \left(8 + \frac{255}{8}\right) = 1432 Bytes$, mientras que antes era de $1472 - (8 + 255) = 1209 Bytes$. Se obtiene una mejora de 223 Bytes.

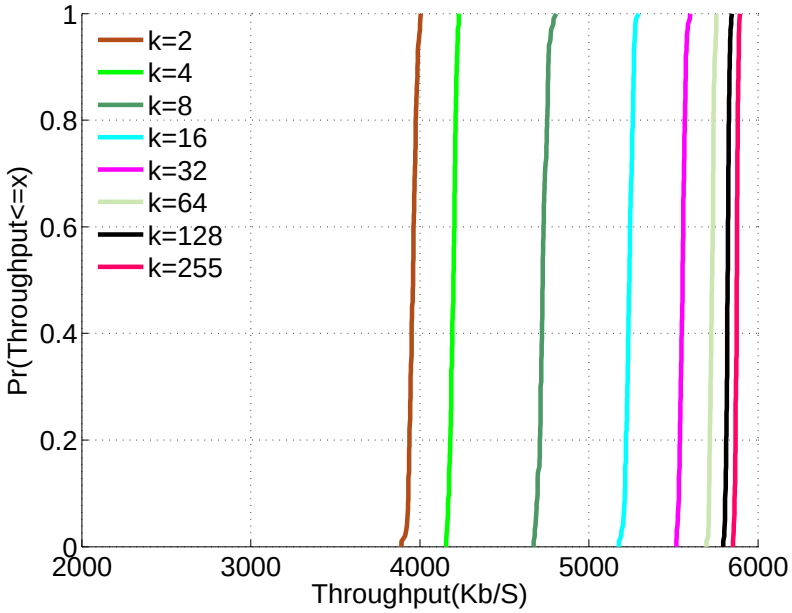


Figura 5.3: Función distribución del throughput para la cabecera final

Se puede apreciar que para la mayoría de los K pequeños no existen grandes diferencias (para $K = 2$ se pruduce una gran mejoría), pero a partir de $K = 32$, el beneficio ya es muy notable. Es por ésto que tras modificar la cabecera propietaria del módulo implementado, se obtuvieron los resultados que aparecen en la Figura 5.3, en la que la mayor ganancia se da con un valor de $K = 255$, y va disminuyendo a medida que K toma valores más pequeños, hasta llegar al mínimo, $K = 2$. Es importante resaltar que también se han mejorado, aunque en menor medida, el rendimiento para los valores de K más bajos.

Otro factor muy importante que se debe de tener en cuenta es que cuando, en el proceso de codificación, se lleva a cabo la generación de los vectores aleatorios, puede darse el caso de que se genere un vector nulo, es decir, con todos sus elementos cero, por lo que no aportaría ninguna información útil, disminuyendo la eficiencia y el rendimiento. Al igual que con la cabecera inicial, al principio también se realizaron experimentos usando los vectores nulos, que estos eran enviados hacia recepción. Este factor también influye en la mejoría que se pone de manifiesto al comparar las Figuras 5.2 y 5.3.

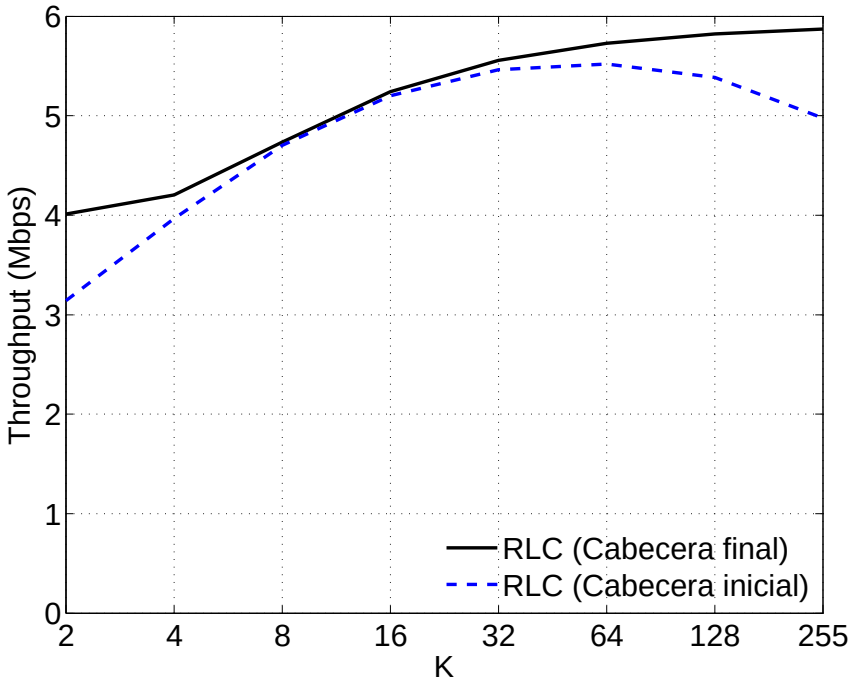


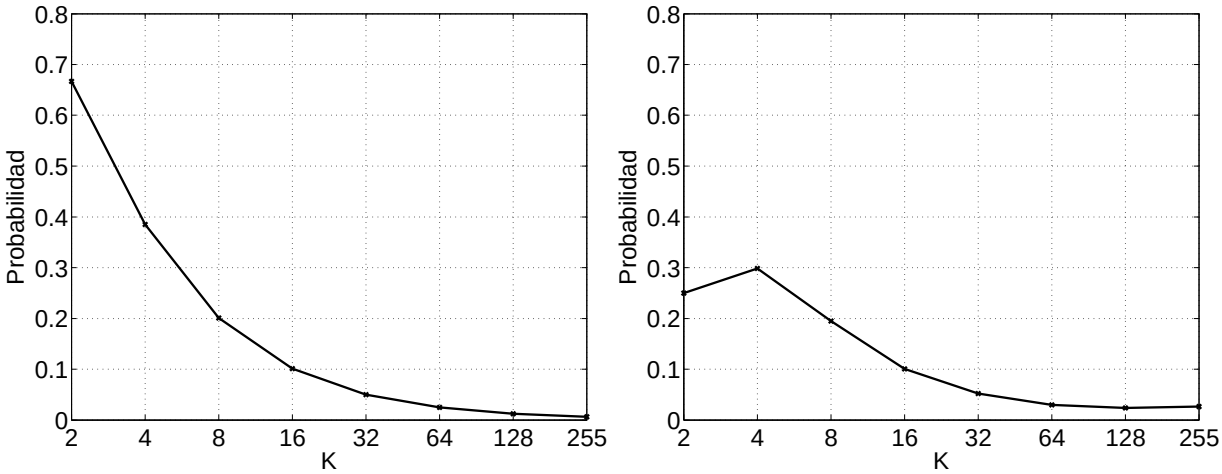
Figura 5.4: Variación del throughput en función de K

A continuación se comprueba de una manera más sencilla, cómo se produce la variación del throughput con respecto a K a medida que ésta aumenta. En la Figura 5.4 se pueden ver ambas curvas, la correspondiente a la cabecera inicial y la que finalmente fue utilizada, que, como ya se ha mencionado, ofrece unos resultados bastante más favorables. Un detalle importante es el hecho que se produce en la curva de la cabecera inicial en torno a los 5500 Kbps ($K = 64$), ya que a partir de ese punto, el throughput comienza a disminuir paulatinamente. Esto es debido a que con la primera implementación que se

realizó, a partir de dicho K, el rendimiento se veía muy penalizado por el gran tamaño que ocupaba la cabecera de *network-coding*.

Número de transmisiones innecesarias

Un factor fundamental a considerar a la hora de estudiar la eficiencia del sistema estudiado es la repercusión que tiene el número de transmisiones, ya que a medida que se aumenta dicho valor, se verá como se emplea más tiempo para finalizar la transmisión y, por lo tanto, se producirá una pérdida de eficiencia. En la Figura 5.5a se puede ver cómo se produce la variación de transmisiones innecesarias en función del valor de K empleado en la codificación; para valores bajos de K se obtienen resultados bastante malos, con unos porcentajes de transmisiones innecesarias entre el 30 % y el 70 %. Esto es debido a que conforme se aumenta la K, la probabilidad de generar un vector aleatorio que sea linealmente independiente es mayor y, por lo tanto, con la K máxima se llega a valores cercanos al 0 %. En un principio se obtuvieron unos resultados bastantes malos (Figura 5.5a), pero una vez introducidas las mejoras en el sistema se puede comprobar, en la Figura 5.5b, la notable mejoría que se produce, especialmente para $K = 2, 4$. Esto es debido a que la eliminación en la transmisión de los vectores aleatorios nulos provoca un gran ahorro en las tranmisiones, y con $K = 2$ es más probable que aparezca un vector nulo (0.25) que con $K = 4$ (0.0625), de ahí que la mejora en la K más baja sea más notable y se produzca ese pico.



(a) Cabecera inicial

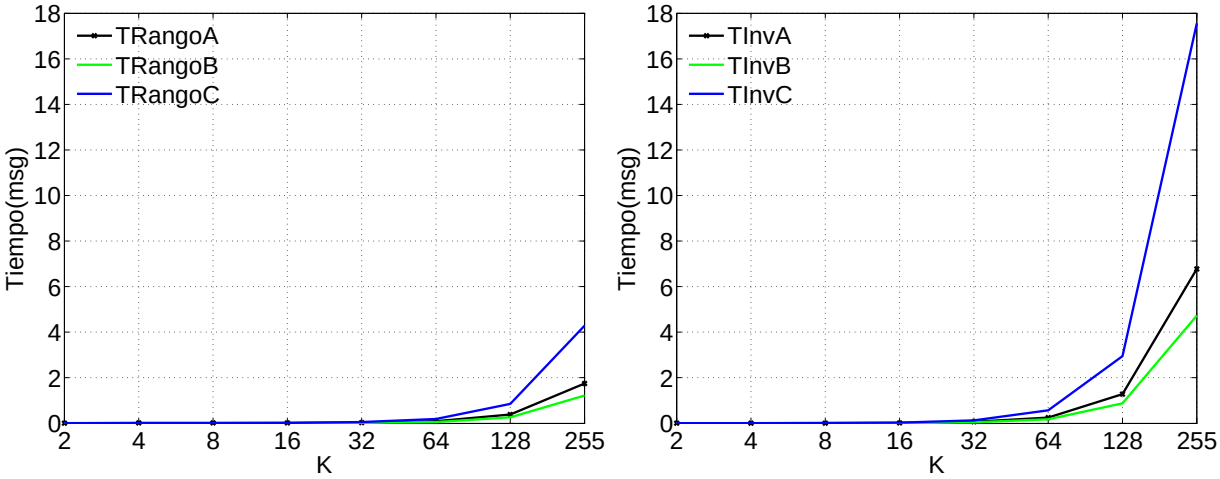
(b) Cabecera final

Figura 5.5: Comparativa de las tranmisiones innecesarias

Tiempos de cómputo

Otro de los parámetros que se ha considerado en este trabajo han sido los tiempos de cálculo, tanto del rango como de la matriz inversa. Se debe de tener en cuenta que estos factores son fundamentales, ya que ambos procedimientos se llevan a cabo en recepción, por lo que un sistema en el que dichos tiempos fueran mayores que el tiempo medio de llegada entre paquetes, se convertiría en un sistema ineficiente, siendo la sobrecarga correspondiente superior al beneficio que aporta.

Para llevar a cabo estas medidas se han utilizado tres equipos diferentes, de manera que se pueda conseguir una mayor variedad en los resultados. En primer lugar se observa la Figura 5.6a, en la que se ve que los tiempos para los diversos equipos, hasta $K = 64$, son prácticamente insignificantes, situándose siempre por debajo del tiempo medio entre las llegadas de paquetes consecutivos; sin embargo, para $K = 128, 255$ se puede comprobar que se empiezan a producir cambios bruscos, especialmente en el equipo C, llegando incluso a sobrepasar los 4 milisegundos, lo que generaría una cola de paquetes en recepción debido al cálculo de los rangos.



(a) Trango (b) Tinversa

Figura 5.6: Tiempos de rango y de matriz inversa

Con respecto a la medida del tiempo empleado para el cálculo de la matriz inversa (Figura 5.6b), cabe destacar que nuevamente se obtienen valores prácticamente nulos en los K bajos, pero a partir de $K = 64$ se pone de manifiesto una fuerte pendiente positiva, alcanzando en el peor de los casos casi los 18 milisegundos, mutiplicando por más de 4 el tiempo empleado en el cálculo de los rangos.

Los equipos utilizados para las simulaciones han sido los siguientes:

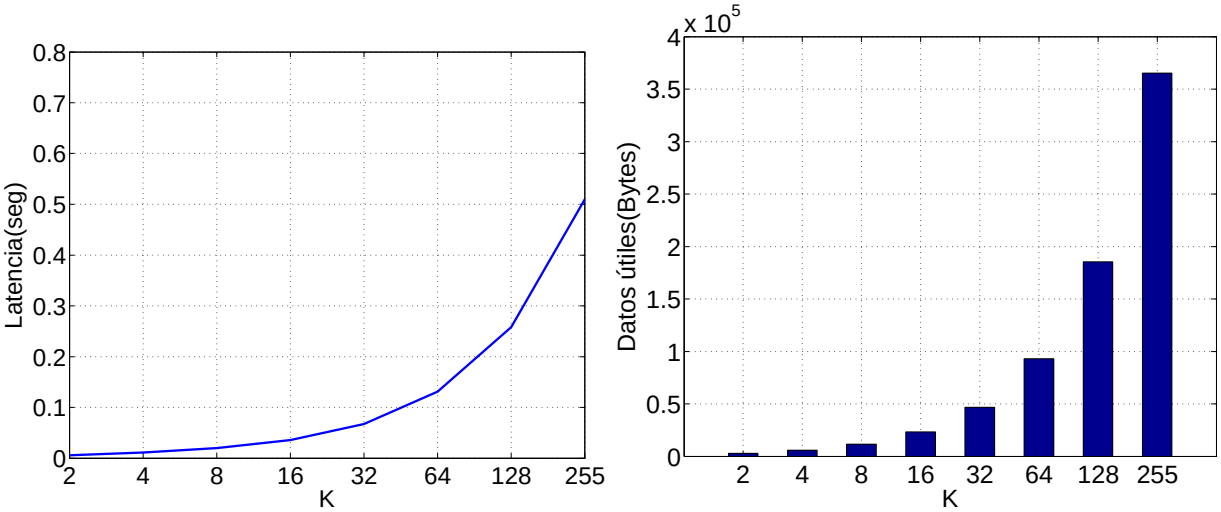
- Equipo A (año 2011), Sony Vaio Intel Core i5-3317-U CPU @ 1.70Ghz 8GB

- Equipo B (año 2012), Lenovo ThinkPad X230 Intel Core i7-3520m @ 2.90Ghz 8GB
- Equipo C (año 2004), Dell Latitude D610 Intel Pentium M @1.86Ghz 2GB

Como se ha podido comprobar en las figuras anteriores, hay una clara y significativa diferencia de un equipo a otro, pasando de uno antiguo y obsoleto (equipo C) a otro intermedio (equipo A) y finalmente el más actual con unas prestaciones mejores (equipo B), que lógicamente ofrece los mejores resultados.

Latencia

En este apartado se va a profundizar en un aspecto fundamental a nivel de usuario, ya que cuando se trabaja con un sistema de comunicación, dependiendo del tipo de datos que se estén transmitiendo, será necesario cumplir unos requisitos en lo que respecta a la eficiencia. En ese sentido no son las mismas prestaciones las que se necesitan en una aplicación de voz IP, donde el retardo entre paquetes es fundamental, que estar realizando la transmisión de un fichero de datos, donde se puede sacrificar el retardo, a cambio de una mejora en el throughput, garantizando que dicho fichero llegue completo.



(a) Latencia

(b) Datos úties

Figura 5.7: Variación del retardo y los datos subidos a aplicación

Una vez dicho esto, es importante analizar la variación del retardo en la Figura 5.7, que aumenta de manera exponencial conforme se incrementa la K. Retomando lo comentado anteriormente, suponiendo que se esté usando voz IP, se tendría que emplear como máximo un valor de $K = 64$ para asegurar una latencia de unos 120 milisegundos, siendo este un valor aceptable para este tipo de servicios. En cuanto al uso de los valores mayores de K,

sería inaceptable, ya que no se pueden permitir retardos de 250 y 500 milisegundos en este tipo de aplicaciones; por el contrario, en el caso de otro tipo de aplicaciones como puede ser la transferencia de ficheros, sería más apropiado emplear un valor de $K = 128, 255$; a pesar de sufrir elevados retardos, se consigue una gran mejora del throughput y además se garantiza la recepción completa de la información.

Comportamiento del throughput en un canal con errores

Hasta este punto, se ha hecho un análisis exhaustivo del sistema sobre un canal ideal; a continuación se llevará a cabo un estudio similar sobre un canal con errores, en el que se producirá la pérdida de paquetes. En la Figura 5.8 se puede comprobar que se produce una disminución constante del throughput a medida que aumenta la FER, llegando hasta una $FER = 0.8$, donde el rendimiento es prácticamente 0 Mbps en el caso de TCP.

Los resultados obtenidos con *network-coding* son bastante mejores que con TCP, pero ambos rendimientos van decreciendo más o menos constante hasta llegar a una $FER = 0.3$, momento en el que TCP sufre una caída bastante brusca. Con el empleo de la codificación el throughput también se ve penalizado, pero de manera menos agresiva, ya que en el peor de los casos se alcanza una velocidad de 0.9 Mbps. En un principio se optó por realizar únicamente medidas hasta una FER máxima de 0.4, pero al comprobar que aún había margen, se tomó la decisión de alcanzar niveles de pérdida mucho más elevados.

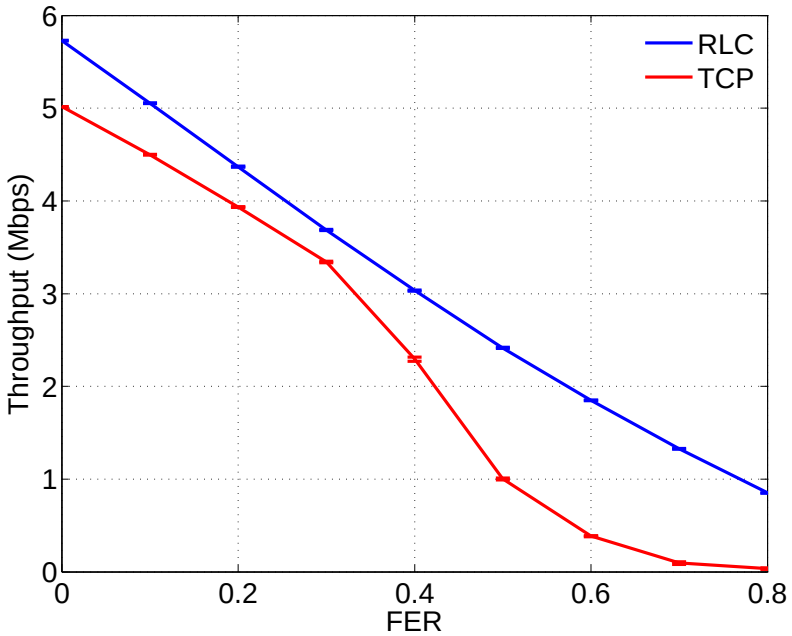


Figura 5.8: Throughput en un canal con errores

Influencia de las retransmisiones 802.11b

Una vez vista la influencia que tiene un canal con errores sobre la transmisión, se va a analizar el efecto que tiene sobre el throughput el hecho de emplear el esquema de retransmisión de 802.11.

El estandar 802.11b cuenta con un mecanismo mediante el cual, si una trama no llega correctamente al receptor y, por lo tanto, no se recibe el ACK de confirmación correspondiente, el transmisor espera un determinado tiempo y nuevamente vuelve a reenviar la trama de datos. A medida que van aumentando las retransmisiones también se produce un aumento del tiempo de espera del transmisor hasta alcanzar un determinado número de retransmisiones, por lo que puede ser muy perjudicial para el sistema, dando lugar a una gran pérdida de eficiencia. En la Figura 5.9 se puede ver como se produce la variación del

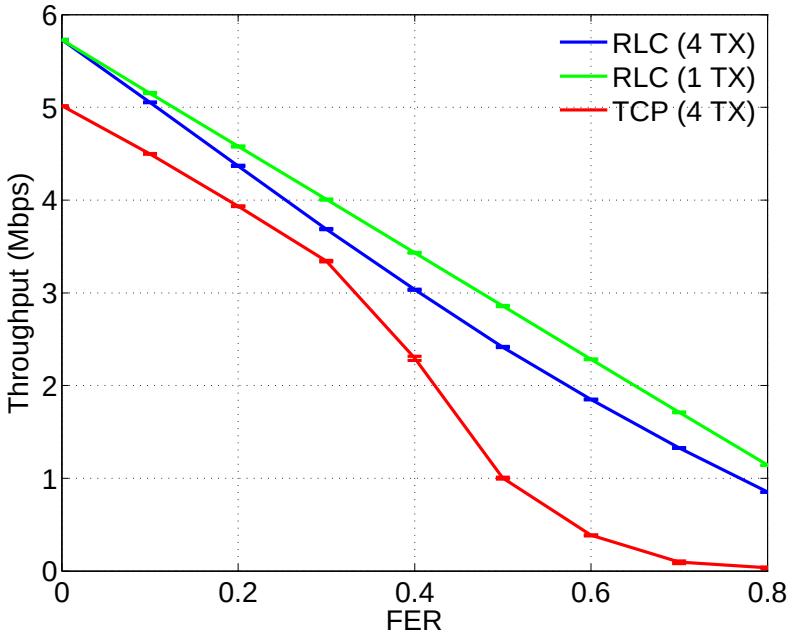


Figura 5.9: Retransmisiones a nivel físico

rendimiento a medida que aumenta la FER. En primer lugar se va a hacer hincapié en los resultados obtenidos realizando únicamente 3 retransmisiones:

- Observando la curva roja (TCP) se ve que, además de obtener los peores resultados, a medida que aumenta la FER y, por consiguiente, las retransmisiones debido a las pérdidas sufridas, se produce un fuerte descenso del throughput.
- Con respecto a la curva azul ($UDP+RLC$), se produce una gran mejoría con respecto a TCP, produciéndose un descenso más constante y sin fuertes bajadas.

Por último se pone de manifiesto que al utilizar $UDP + RLC$, es más apropiado no realizar ninguna retransmisión, por lo que la información se envía únicamente una vez, llegue o no. Si se analiza la situación, es sencillo percatarse de que esta es sin lugar a duda la mejor opción, ya que al enviar la información codificada, el hecho de que se pierda un paquete no es relevante; es decir, todos los paquetes proporcionan la misma cantidad de información. Sería una pérdida de rendimiento realizar retransmisiones, y además no aportaría ninguna ventaja.

6

Conclusiones y líneas futuras

En este último capítulo se recogen las principales conclusiones que se han ido recopilando durante la realización del proyecto, haciendo especial hincapié en aquellas obtenidas tras analizar los resultados del capítulo anterior. Finalmente se comentarán posibles líneas futuras de investigación que han quedado abiertas a raíz del trabajo llevado a cabo.

6.1 Conclusiones y líneas futuras

En este trabajo se ha presentado el diseño e implementación de un nuevo esquema que se sitúa entre las capas de red y transporte. Su objetivo principal es hacer frente a los objetivos básicos de TCP, con el fin de solucionar los inconvenientes que este protocolo presenta sobre redes inalámbricas, donde su rendimiento se ve muy comprometido, especialmente bajo condiciones hostiles. La propuesta que se ha hecho ha combinado el funcionamiento de un protocolo de transporte (UDP) con una codificación basada en RLC.

Por medio de simulaciones (NS-3) y un modelo de análisis, se ha conseguido demostrar que la solución empleada es capaz de mejorar un 15 % el rendimiento alcanzado por TCP sobre enlaces inalámbricos sin errores, obteniendo una ganancia de hasta 1100 % cuando las condiciones del canal son hostiles ($FER = 0.8$). La solución se basa en una codificación llevaba a cabo sobre campos finitos GF(2).

Una vez que se han demostrado los beneficios de esta propuesta sobre una topología de red sencilla, se debe tratar de ampliar el análisis a escenarios más complejos. En particular, las siguientes líneas de investigación se estudiarán en el futuro próximo:

- La primera, y más simple, consistiría en incrementar el uso del *network-coding*, de manera que algunos de los nodos intermedios pudieran hacer recodificaciones linealmende independientes de los paquetes recibidos, con el objetivo de alcanzar mejoras en el rendimiento.

- Por otra parte, se debería ajustar de manera dinámica el valor de K, de acuerdo con los requisitos de las diversas aplicaciones. Por ejemplo, para restricciones de tiempo real, K se podría adoptar de acuerdo a las condiciones del canal.
- Otro aspecto interesante a analizar sería afrontar un estudio del rendimiento obtenido usando campos finitos más grandes. Esto probablemente reduciría la probabilidad de tener vectores linealmente dependientes, y se podría incrementar el rendimiento del sistema para el mismo K. Por otra parte, también tendría un impacto sobre el tamaño de la cabecera y además habría que tener en cuenta el coste de dichos procesos de cálculo adicionales.
- Por último, pero no menos importante, sería buena idea evaluar el impacto de la solución implementada sobre modelos de canal inalámbrico más realistas.

Bibliografía

[1] R. Ahlswede et al. "Network Information Flow". IEEE Transactions on Information Theory, Vol.46, No.4, July 2000.

[2] Sachin Katti et al. "XORs in The Air: Practical Wireless Network Codingïn SIGCOMM'06, Pisa, Italy, September 2006.

[3] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. "MORE: A network Coding Approach to Opportunistic Routing". MIT-CSAIL-TR-2006-0049, June 2006.

[4] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. "Trading Structure for Randomness in Wireless Opportunistic Routing"MIT-CSAIL-TR-2007-014, February 2007.

[5] 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.

[6] 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.

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

[8] 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.

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

Lista de acrónimos

NS-3 Network Simulator 3

WMN Wireless Mesh Network

UDP User Datagram Protocol

TCP Transmission Control Protocol

QoS Quality of Service

WLAN Wireless Local Area Network

DSL Digital Subscriber Line

ACK Acknowledgement

ETX Embedded Technology eXtended

FER Frame Error Rate

NC Network Coding