ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

DESPLIEGUE DE UNA PLATAFORMA PARA ANÁLISIS DE TÉCNICAS DE NETWORK CODING UTILIZANDO RASPBERRY PI

(Deployment of a platform for analysis of network techniques coding using Raspberry Pi)

Para acceder al Titulo de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Fátima Fernández Pérez

Oct.- 2017

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

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Fátima Fernández Pérez
Director del TFG: Ramón Agüero Calvo y Pablo Garrido Ortiz
Título: "Despliegue de una plataforma para análisis de técnicas de Network Coding utilizando Raspberry Pi "
Title: "Deployment of a platform for analysis of network techniques coding using Raspberry Pi"

Presentado a examen el día: 25/10/2017

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): Mañana Canteli, Mario
Secretario (Apellidos, Nombre): Agüero Calvo, Ramón
Vocal (Apellidos, Nombre): García Arranz, Marta

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

Fdo.: El Presidente

Fdo.: El Secretario

Fdo.: El Vocal

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

Vº Bº del Subdirector

Trabajo Fin de Grado Nº
(a asignar por Secretaría)

Índice

Índice de figuras

Índice de tablas

Resumen ejecutivo

En este documento se presenta un estudio sobre la implementación de técnicas de Network Coding (NC) en un entorno real, sobre una red multi-salto. En concreto se estudiará el esquema Random Linear Network Coding (RLNC), que es una de las principales alternativas de NC. Esta solución permite la transmisión de información codificada entre dos nodos de la red.

El continuo crecimiento en la utilización de redes inalámbricas, ha despertado un gran interés por parte del sector de la investigación, buscando posibles alternativas a los protocolos tradicionales, debido a que éstos pueden no dar una respuesta adecuada a las condiciones adversas del canal radio. La solución que se va a estudiar es NC, que implementándose sobre el protocolo de capa de transporte User Datagram Protocol (UDP), responde de forma eficiente, a las adversidades que se puedan presentar, principalmente sobre canales inalámbricos.

Para poder discutir los resultados obtenidos mediante simulación, en este proyecto se ha llevado a cabo una implementación en entorno real de dichas técnicas, en concreto, en una plataforma compuesta por 20 nodos, los cuales se corresponden a Raspberry Pi 3 modelo B, formando una red multi-salto para la configuración de distintas rutas.

Abstract

In this document we present a study on the implementation of Network Coding (NC) techniques in a real environment, over a multi-hop network. Specifically, we study the Random Linear Network Coding (RLNC) scheme, which is considered one of the most relevant NC solutions. This scheme allows the comunication between two nodes in the network.

Wireless networks have received a great interest from the research community, searching for posible alternatives to traditional protocols which could not give an appropriate response to adverse channel conditions. One of the alternatives NC, which it is implemented over the transport layer UDP protocol, and to other a reliable service it is to believed to address hostile conditions.

In order to discuss the results obtained by simulation, this project has worked on the implementation of NC techniques in a real environment, specifically, in a platform comprising twenty nodes, Raspberry Pi 3 model B, building a multi-hop network where different routes can be configured.

Agradecimientos

Quiero agradecer, en primer lugar, a Ramón Agüero y a Pablo Garrido por su ayuda y paciencia, que se que no es poca, a la hora de realizar este proyecto. Muchas gracias.

También agradecer a mi familia, en particular a mis padres y a mi hermana por el apoyo incondicional en estos cuatro años. África, ni a la hora de acabar la carrera nos diferenciamos. En referencia a los que se han marchado en esta etapa, gracias por verme crecer.

Por último, mi agradecimiento a mis amigos por las necesarias horas de desconexión y entretenimiento. Y a todos los que me he encontrado en estos años, que de alguna u otra manera estáis dejando huella.

1 Introducción

En este primer capítulo se explicará la motivación de llevar a cabo este trabajo, así como los diferentes objetivos que se quieran alcanzar. Finalmente, se detalla la estructura que sigue el documento, con el fin de adelantar y entender los diferentes elementos que se tratan en la memoria.

1.1. Planteamiento del problema

El incremento en número de dispositivos conectados se ha producido en parte gracias a la aparición de las redes inalámbricas. Este éxito ha tenido como consecuencia una constante evolución de dichas redes; desde el tipo de seguridad hasta conseguir la máxima efectividad en su modo de operación. Hasta nuestros días, se han ofrecido una variedad de prestaciones mejoradas de las redes inalámbricas.

Con el paso del tiempo se ha visto que las redes inalámbricas son fundamentales para la conectividad entre usuarios. Resaltar que el protocolo más utilizado en la capa de transporte es Transmission Control Protocol (TCP), usado en diferentes aplicaciones, y que ofrece fiabilidad extremo a extremo y entrega ordenada en la comunicación. Otra gran alternativa a nivel de transporte es User Data Protocol (UDP). UDP no ofrece una comunicación fiable extremo a extremo, pero es utilizado por aplicaciones con altos requisitos de throughput y bajo retardo, por ejemplo, streaming de vídeo, donde la pérdida de un paquete pudiera no ser tan importante.

TCP fue diseñado para redes cableadas, donde la probabilidad de producirse un error en la transmisión de una trama es muy baja. Como es conocido, TCP no tiene un buen comportamiento en redes inalámbricas, que se caracterizan por ser propensas a errores debido a unas condiciones más adversas a nivel de interferencias y ruido. El control de la congestión que ofrece este protocolo no distingue cuando una trama se ha perdido debido a la congestión de la red o a interferencias en el canal, perjudicando gravemente el rendimiento de TCP sobre este tipo de redes. Hay que tener en cuenta, además, que el canal radio puede ser inestable, considerando que el usuario pueda moverse y como consecuencia, una alta variabilidad del tiempo de transmisión. La alternativa que se estudia en este proyecto son las técnicas de Network Coding (NC), en concreto el esquema RLNC. Esta técnica de NC utiliza a nivel de transporte el protocolo UDP con la característica de que ofrece un servicio fiable como TCP, pero el número de reconocimientos transmitidos por parte del nodo destino pudiendo conseguir que la red no se congestione, con lo que se mejora la eficiencia de esta y se consigue un ahorro energético.

En definitiva, en este trabajo se propone un nuevo enfoque para garantizar una transmisión fiable de extremo a extremo sobre redes multi-salto donde los nodos son dispositivos de bajo coste y baja capacidad computacional.

1.2. Objetivos

Son muchos los estudios que existen hasta el momento sobre el uso de técnicas de NC y sus distintas variantes. Casi todos los estudios se han aplicado en entornos teóricos y es interesante su estudio en un despliegue real.

En este proyecto se desplegará un banco de medidas sobre el que se analizará, bajo experimentación, el uso de técnicas NC. Estos resultados se compararán con los teóricos. El despliegue consiste en una red mallada compuesta por 20 nodos, Raspberry Pi, y se configurará como una red multisalto donde se harán pruebas de rendimiento en función de los saltos intermedios entre el transmisor y receptor, usando las distintas técnicas de Network Coding y los distintos protocolos de la capa de transporte.

Se usarán 20 nodos para que la recogida de medidas sea más completa, ya que se podrán probar distintas configuraciones en la red, que cuenta con cuatro transmisores y cuatro receptores pudiendo obtener cuatro rutas distintas. También se modificarán los distintos parámetros que hay que aportar para las técnicas de Network Coding.

Para el despliegue de la plataforma y su configuración se utilizará un código propietario en C++. Para el esquema de codificación se hace uso de la librería Kodo. Y, además, para facilitar el uso y la puesta en marcha de la plataforma se utilizan script en Python.

1.3. Estructura de la memoria

A continuación se detalla la estructura del documento, revisando brevemente el contenido de cada uno de los capítulos.

- En el Capítulo 2 se presentan los conceptos previos necesarios para desarrollar y entender el trabajo. Se explican los aspectos teóricos como red multi-salto, escenario en el cual se llevará a cabo el proyecto y en el que se estudiarán los distintos protocolos de transporte que se usan en la transferencia de datos, además de probar las técnicas de NC. También se hace referencia a los diferentes trabajos existentes centrados en el estudio de las técnicas de Network Coding. Se aprovechará para definir conceptos fundamentales para entender este proyecto y se explicará qué es NC y cómo se ha implementado esta técnica a través de la librería Kodo.

- El Capítulo 3 supone la parte central del trabajo. En primer lugar se presentan los materiales que han sido necesarios para llevar a cabo este proyecto, A continuación se detalla cómo se ha realizado el escenario de trabajo y las herramientas para la obtención de medidas.

- En el Capítulo 4, se obtienen los resultados a partir del banco de medidas, donde se medirá el throughput en los distintos casos: protocolos de transporte TCP y UDP y el uso de la técnica RLNC, comparando los resultados de los tres esquemas de medida. Se hará una comparación final utilizando otra técnica de NC, TSNC.

- Por último, en el Capítulo 5, se detallan las conclusiones que se han alcanzado tras el análisis de los resultados obtenidos previamente y la explicación de las líneas futuras del uso de este esquema de comunicación.

2

Estado del Arte

Este capítulo recoge una introducción de cada uno de los conceptos en los que se apoya el trabajo. De esta forma el lector puede adquirir unos conocimientos básicos que le permitan comprender los diferentes conceptos tratados a lo largo del proyecto.

2.1. Redes Multi-salto

Una red multi-salto o red *Ad-hoc* [1], se puede definir como un conjunto de nodos autónomos que se agrupan de manera dinámica sin la necesidad de que haya un nodo central para controlarlos. Es una red con una topología descentralizada donde no existe un punto de acceso actuando cada nodo como enrutador y generador de tráfico. Las comunicaciones entre los nodos se realizan a través de conexiones inalámbricas con lo que comparten el canal radio, lo cual puede hacer que el rendimiento de la red disminuya.

Debido a que los nodos se pueden mover por la red y comunicarse entre ellos de manera aleatoria la topología varía sin poder hacer una previsión de sus características. En las redes multi-salto puede haber más de un nodo intermedio entre un emisor y otro receptor, lo cual da aun más incertidumbre al comportamiento de la red.

Hay diversas formas de clasificar los protocolos de enrutamiento de las redes multi-salto, siendo la más común su modo funcional, pudiendo ser *preventivos* o *reactivos*.

Los protocolos *preventivos* se basan en que los terminales disponen en todo momento de la información sobre la topología de la red. Esta información se transmite a todos los nodos a través de mensajes de control. Por otro lado, en los protocolos *reactivos*, los nodos buscan una ruta para comunicarse con el resto bajo demanda, es decir, cuando la necesitan. También se pueden clasificar dichos protocolos mediante la función que desempeñan los nodos de la red o de la estructura de la red Ad-hoc. De otra manera, también se pueden agrupar atendiendo al consumo de energía, los que habilitan comunicaciones multicast o los que se basan en información geográfica. En este proyecto, se utilizarán tablas estáticas. Los nodos de la red tendrán configuradas las tablas de rutas sin la necesidad de implementar estos protocolos de enrutamiento, evitando así el impacto que tendrían en el comportamiento de la red, de forma que sea más fácil comprender los resultados que se obtienen.

La red multi-salto se utilizará para analizar el rendimiento en función de los nodos que ha de atravesar la información desde un origen a un destino. Se debe tener en cuenta que, al tener varios dispositivos, el número de saltos puede variar, haciendo que el rendimiento pueda disminuir notablemente. Por cada salto más en la red, el rendimiento disminuiría. Otros trabajos [8] señalan el problema que supone el uso del protocolo TCP sobre este tipo de redes ya que fue diseñado para redes cableadas, donde la probabilidad de error es despreciable y la congestión es la principal causa de la pérdida de paquetes.

Adicionalmente, las redes multi-salto son apropiadas para las redes de sensores, debido a que estos componen una topología que constituye una red mallada. Con esto se entiende que las redes malladas (Wireless Mesh Network (WMN)) son un tipo de redes Ad-hoc que permiten ofrecer de forma fácil, y económica, un mayor área de cobertura.

Hasta ahora, en el Grupo de Ingeniería de Telemática no se ha caracterizado en profundidad las redes multi-salto usando técnicas de Network Coding en un entorno real.

2.2. Network Coding

Los nodos de la red almacenan y retransmiten la información que circula a través de los mismos sin ningún tipo de procesado, basándose en sus tablas de rutas, sin ningún procesado de ésta.

En contra posición aparecen las técnicas de codificación de red, *Network Coding (NC)* de la mano de *Ahlswede, Cai, Li y Yeung* [2], donde los enrutadores son capaces de modificar los paquetes mediante técnicas de codificación, entendiendo codificación como la combinación de 'fragmentos' de la información a transmitir, permitiendo gestionar los recursos de forma inteligente. Desde entonces, se ha estudiado el uso de estas técnicas sobre diferentes tipos de redes, en concreto en las redes inalámbricas multi-salto. Además, después de varios estudios [3], se establecen dos líneas de investigación, una donde se combinan los

paquetes pertenecientes a diferentes flujos de información provenientes de distintos nodos intermedios, *inter-flujo*. Otra en la cual se potencia el uso de combinaciones aleatorias lineales de paquetes pertenecientes al mismo flujo, es decir, no combina la información transmitida de distintos nodos, *intra-flujo*. En este segundo grupo es en el que se va a centrar este proyecto.

Las técnicas de Network Coding se implementarían por encima de la capa de red (Internet Protocol (IP)), a nivel de capa de transporte.

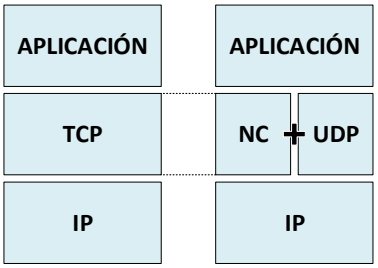


Figura 2.1: Cambio en la arquitectura tradicional con Network Coding

El hecho de que estas técnicas provoque la reducción de transmisiones hace que el dispositivo tenga un mayor ahorro energético, manteniendo un servicio fiable e introduce nuevos mecanismos de seguridad debido a que se podría decir que la información va 'encriptada'.

Aunque en este proyecto no se vaya a utilizar de manera directa, cabe destacar que a pesar de todas las características que tiene la codificación de red, una de las principales ventajas es la característica broadcast del medio radio, ya que las comunicaciones inalámbricas seguían enfocadas a transmitir en modo unicast. Debido a esto, los demás nodos vecinos del destinatario pueden recibir y almacenar la información mediante una escucha oportunista.

Con la evolución de esta técnica y los estudios que se están llevando a cabo se espera que tras constatar su validez y caracterizarse en un entorno real, NC pueda convertirse en un componente más de redes malladas.

A continuación, se explicarán conceptos básicos que se utilizarán a lo largo del trabajo para entender en qué consiste las técnicas de *Network Coding*.

2.3. Inter-flujo Network Coding

Si bien el esquema de *inter-flujo* no es en el que está basado este proyecto, conviene explicarlo superficialmente para que se tenga una comprensión completa de las técnicas de codificación, y así entender la diferencia con el esquema de *intra-flujo*.

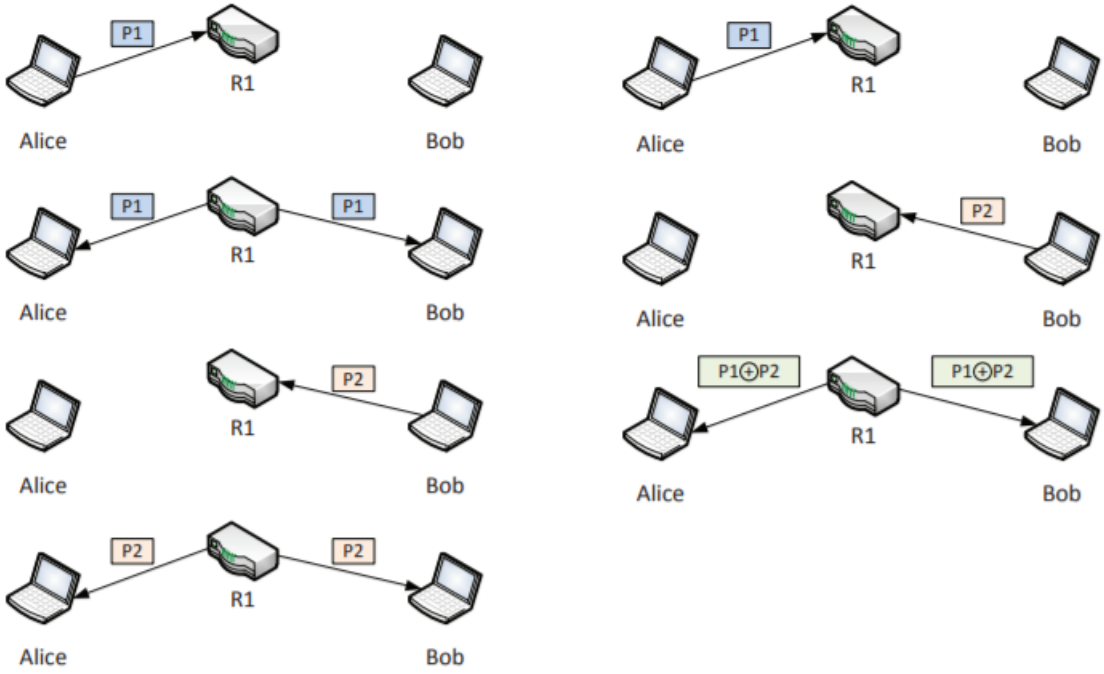
Este método está basado en una versión simplificada de uno de los estudios más importantes en Network Coding como es Opportunistic Coding in Practical Wireless Network Environment (COPE) [4]. Según esta propuesta, un nodo, que pasará a ser un *coding node* (C_N), interviene en la transmisión de dos flujos de datos independientes, dentro de su área de cobertura, mediante la técnica de 'escucha oportunista' y los almacena en su buffer. Al recibir los paquetes, el *coding node* realiza una combinación de ellos y procede a enviarla en modo broadcast a los nodos de la red. Los receptores de dicha transmisión se encargarán de decodificar, utilizando la información que ya conocen, el flujo de datos que les envía el *coding node*.

El funcionamiento de esta técnica tiene que tener en cuenta un tiempo denominado *Coding Time* (C_T). Es importante, ya que como no se tiene la seguridad de que dos paquetes puedan llegar 'simultaneamente' al C_N para que este los codifique, debido a la aleatoriedad de este tipo de redes. Este parámetro representa el tiempo máximo que un paquete puede estar almacenado en el nodo sin combinarlo con otros. El nodo espera C_T a que lleguen más paquetes, si en ese tiempo no los recibe, retransmite lo que tenga en el buffer sin codificarlo. Mediante esta función se pueden evitar las congestiones en la red por largas esperas.

Al parámetro C_T se le puede añadir otro de suma importancia, como es el *Buffer Size* (B_S). Su cometido es determinar la cantidad de paquetes que se pueden almacenar en el C_N a la espera de una oportunidad de codificación. Con esto se pretende aumentar las posibilidades de que los paquetes puedan estar codificados.

La forma de codificar del C_N consiste en coger los paquetes y realizar una sencilla operación XOR a nivel de bit. Así, en los nodos receptores, cuando les llegue el paquete codificado, con los paquetes nativos que tengan en su buffer, podrán decodificar la información que les interesa, aplicando la misma operación. Resaltar el detalle de que el receptor solo podrá decodificar si, y solo si, tiene información nativa del paquete. La ventaja de este método es que refuerza notablemente la seguridad de la comunicación, debido a que el receptor tiene que tener unos conocimientos previos de la información que le va a llegar para decodificarla: conocer los paquetes nativos y el tipo de decodificación además del paquete a analizar.

Para que el proceso quede más explícito, en el ejemplo de la Figura 2.2 [5] se puede apreciar el mecanismo explicado anteriormente. En la imagen (a) se muestra lo que sería la comunicación más tradicional TCP: Alice le envía el paquete, P_1, a R1. A continuación R1



(a) Escenario sin NC

(b) Escenario con NC

Figura 2.2: Escenario sin NC v. escenario con NC

se lo transmite a Bob, ya que él era el destinatario. Bob le envía el paquete, P_2 a R1 para que esté se lo envíe a Alice. De esta manera se han realizado cuatro transmisiones, lo que equivale a más gasto energético y a un gasto de tiempo que se ahorra en el siguiente ejemplo, donde se usa NC. En la imagen(b) ya se pone en práctica el método *inter-flujo*. Alice y Bob transmiten a R1, en distinto espacio temporal, los paquetes que se quieren enviar, P_1 y P_2. R1 se encarga de combinarlos, creando el paquete que emitirá ($P_c = P_1 \oplus P_2$) a Alice y a Bob, que se encuentran dentro de su rango de cobertura. Los receptores, mediante sus paquetes nativos, podrán decodificar la información que deseen, es decir, Alice decodificará P_2 mediante una XOR; $P_2 = P_c \oplus P_1$, y Bob se quedará con el paquete P_1, $P_1 = P_c \oplus P_2$.

Así, una comunicación que normalmente requiere de cuatro transmisiones, se puede realizar solo con tres, haciendo uso de Network Coding. Se ha mejorado la eficiencia energética (menos retransmisiones) y se han reducido los retardos, los recursos ocupados y se evita la congestión de la red.

2.4. Intra-flujo Network Coding

El enfoque de esta técnica, que es en la que se basa el proyecto, es distinto al explicado en la Sección 2.3, específicamente en el esquema RLNC. Se procederá a explicar el

funcionamiento del esquema RLNC y las características que se han aprovechado en este trabajo.

Dentro de la disciplina de NC se puede realizar un tratamiento de la información distinta, cuando pertenecen a flujos independientes (*Inter-flow*) o que provenga de un mismo flujo como es en este caso (*Intra-flujo*). La principal aplicabilidad de este método es en las redes de conmutación de paquetes. La primera vez que se planteó este esquema fue mediante el estudio MAC-independent opportunistic routing protocol (MORE) en [6] donde ya se planteaba el hecho de que los routers combinaran la información antes de transmitirla.

En el esquema de *Intra-flujo* se usa como protocolo de la capa de transporte UDP, y combinaciones lineales aleatorias sobre los datagramas correspondientes, antes de su entrega. Los nodos fuentes se encargan de llevar el proceso de codificación de los paquetes que llegan y los almacenará en su buffer de transmisión. Este protocolo difiere de lo visto en la Sección 2.3 que no combina flujos independientes de información, por lo que a cada flujo le asigna un 'sub-buffer', es decir, los que tengan el mismo identificador.

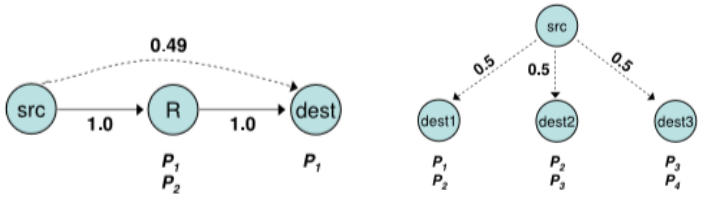
Antes de seguir con la explicación de este protocolo es conveniente definir unos conceptos previos para que no haya posibilidad de error en la interpretación:

Tabla 2.1: Definición de conceptos.

Término	Definición
Paquete nativo	Paquete sin haber pasado por el proceso de codificación.
Paquete codificado	Paquete que ha sufrido mecanismos de codificación.
Paquete innovador	Paquete creado por una combinación linealmente independiente a los recibidos con anterioridad.
Generación	Bloque de paquetes en los que se dividirá la información total.
K	Número de paquetes de los que constará una generación.
GF(Q)	Cuerpo finito a partir del cual se generan los coeficientes.
Vector de codificación de un paquete codificado	El vector que contiene los coeficientes utilizados en la combinación de los paquetes.

El funcionamiento básico de este método consiste en que hay un nodo origen, que quiere transmitir a un nodo destino. La información a enviar serán los K paquetes en los que se trocea la información, que son de longitud fija (paquetes nativos). A continuación, el codificador genera una combinación lineal de los K paquetes nativos, dando como resultado un paquete codificado. Una vez recibido, el nodo destino lo almacena. El origen transmite tantos paquetes codificados como sean necesarios para que el destino sea capaz de decodificar la información inicial. El receptor almacena cada paquete recibido en una matriz,

en la que cada fila es linealmente independiente a la anterior. El nodo destino conoce los coeficientes utilizados en cada paquete codificado, gracias al vector de codificación que se transmite junto a ellos, y que permite que la información pueda ser decodificada.



(a) Ejemplo unicast (b) Ejemplo multicast

Figura 2.3: Esquemas Intra-flujo

Para que quede una explicación clara se procederá a explicar dos casos concretos que se presentan en [6]. El primer ejemplo es en un escenario unicast, Figura 2.3a , que consta de dos nodos, origen (src) y destino (dst), y un router intermedio (R). src quiere comunicarse con dst enviándole dos paquetes, P_1 y P_2, a través de R, pero el destino escucha uno de los dos paquetes. R no sabe qué paquete es el que ha podido obtener el destino, en consecuencia tendría que transmitir tanto P_1 como P_2, lo que produciría una comunicación poco eficiente. NC ofrece una solución sencilla que consiste en que R combina P_1 y P_2 para conseguir la transmisión de un solo paquete ($P_c = P_1 + P_2$). Así, el destino puede completar la información con la que ya obtuvo. Generalizando aun más el esquema, los enrutadores pueden generar combinaciones mediante coeficientes aleatorios (C_i), es decir, el origen crearía ($P_c = C_1 \cdot P_1 + C_2 \cdot P_2 + ... + C_i \cdot P_i$). El destino enviaría un ACK una vez hubiera obtenido toda la transferencia.

En segunda instancia, en la Figura 2.3b, se muestra un ejemplo que recoge el caso multicast. La fuente quiere transmitir a tres destinos los paquetes P_1, P_2, P_3 y P_4. Se supone que el primer destino recibe P_1 y P_2, el segundo P_2 y P_3 y el tercero P_3 y P_4. Como los enlaces tiene una probabilidad de 0.5, la mitad de los paquetes que se envían se perderán, por lo tanto ningún nodo destino habrá recibido todos los paquetes y el origen tendría que retransmitir otra vez la información hasta cuatro veces. Usando NC el origen con solo transmitir dos paquetes se completarían todas las recepciones. Es decir, el transmisor puede emitir: $P'_1 = P_1 + P_2 + P_3 + P_4$ y $P'_2 = P_1 + 2P_2 + 3P_3 + 4P_4$, así los destinos serían capaces de reconstruir la información original.

Para recuperar la información que ha enviado el nodo origen los nodos destinos tienen que resolver un sistema de ecuaciones, para lo que el decodificador utiliza eliminación Gaussiana.

2.4.1. Random Linear Network Coding

El método que se utiliza en este proyecto es el llamado *Random Linear Network Coding*, (RLNC) que fue propuesto por primera vez en [7]. Consiste en una combinación del protocolo de transporte UDP y la combinación lineal de los paquetes de un mismo flujo de comunicación. Con los conceptos recogidos en la Tabla 2.1, [8] y [3], se procederá a explicar el protocolo.

Los paquetes se agruparán en diferentes buffers, según el flujo al que pertenezcan (los que comparten el mismo ID de flujo se almacenarán juntos). Cuando se han almacenado K paquetes nativos del mismo flujo, se crea una combinación lineal aleatoria de ellos (de ahora en adelante, las combinaciones realizadas sobre el mismo grupo de paquetes nativos se denominarán 'generación'), y un paquete codificado:

$$p' = \sum_{i=0}^{K-1} p_i \cdot c_i \tag{2.1}$$

En la Ecuación 2.1 se expresa la operación para crear un paquete codificado donde c_i son los coeficientes aleatorios generados a partir del cuerpo finito GF(Q) = GF (2^q) y los p_i's son los paquetes nativos. Los coeficientes aleatorios se pueden representar como un vector de codificación, c = [c_0, c_1, c_2,..., c_{K-1}]. La fuente enviará periódicamente paquetes codificados. El nodo destino tendrá dos formas de proceder, una donde confirme, enviando un Acknowledgement (ACK), la recepción total satisfactoria de la generación y otra en la que no confirmará que se ha recibido la generación.

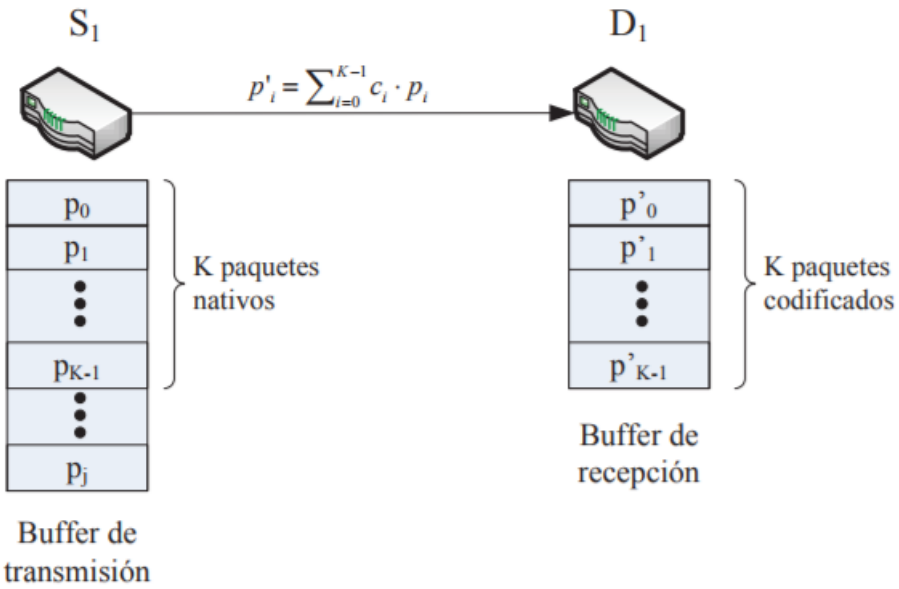


Figura 2.4: Escenario RLNC

Con el fin de no sobrecargar los buffers de capa inferior del transmisor y no provocar una congestión, el protocolo inyecta paquetes dinámicamente, de acuerdo con la tasa de salida física.

Para el proceso de decodificación el nodo destino usa dos entidades de almacenamiento: una matriz cuadrada C de rango K, que almacena los vectores codificados en sus filas y un buffer para almacenar los paquetes codificados. Cuando llega un paquete codificado, p_i', su vector de coeficientes se añadirá a la j-ésima fila de la matriz C. Si dicho vector es linealmente independiente de los anteriores, se traduce en que el rango de la matriz aumenta, el paquete codificado se traducirá a información útil y se almacenará en el buffer de recepción. Si el vector codificado es linealmente dependiente, entonces se eliminará de C y el paquete se descartará. El receptor necesita almacenar K paquetes nuevos con información útil para poder decodificar la generación correspondiente. Esto significa que cuando se descarta un paquete no hay mayor inconveniente porque es fácilmente sustituible por otro útil.

Tras la recepción de K paquetes con vectores de codificación linealmente independientes, la matriz C será de rango K y se podrá calcular su inversa C^{-1}. A partir de la matriz inversa se podrán recuperar los paquetes nativos, mediante la operación P = C^{-1}·P'. Una vez que el destino ha recibido los K paquetes nativos y los ha decodificado actúa dependiendo de como esté configurado como se ha dicho anteriormente: o envíando un ACK al nodo origen confirmando así la correcta recepción de la información, para que pase a enviar la siguiente generación. Además procurará, si es posible, borrar las tramas que contienen los paquetes codificados que se encuentran en el buffer de la capa IEEE 802.11 MAC. La Figura 2.4 ayuda a entender más explícitamente este esquema de codificación. En otras configuraciones, el receptor podría no enviar el ACK (multicast).

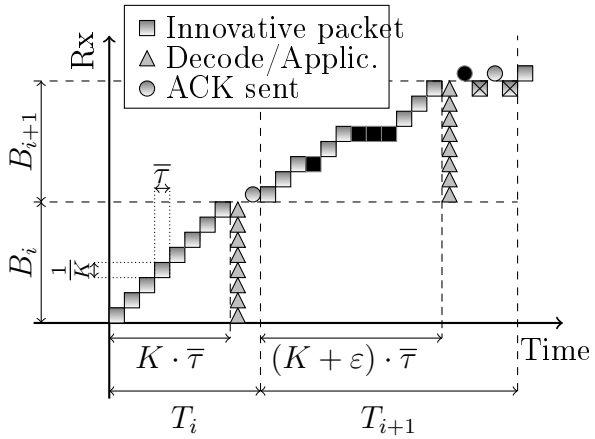


Figura 2.5: Paquetes en el receptor con K=8

En la Figura 2.5 se representa la recepción de dos paquetes con K=8. Al inicio se ve que se ha recibido un paquete corréctamente (F_i), cuando el receptor ha recibido K paquetes

linealmente independientes y puede calcular la matriz inversa de C, como se ha explicado anteriormente, para obtener la información. En el segundo fragmento se reciben paquetes que no son linealmente independientes, los cuales son descartados de forma automática. En consecuencia, se necesitarán $K + \beta$, siendo β el número de transmisiones adicionales necesarias para poder completar la generación. Finalmente se manda un ACK al nodo origen para confirmar la correcta llegada de la generación. En el hipotético caso de que el ACK no llegue al transmisor este seguirá enviando paquetes y, en consecuencia, el receptor los descartará y volverá a mandar un ACK.

Los paquetes pueden llegar al destino en cualquier orden, ya que a la hora de decodificarse se almacenará la información en el orden correcto. Además, si un paquete se extravía no supone una pérdida relevante, debido a que todos contienen la misma información. En consecuencia, la codificación de los datos aporta más seguridad y mayor robustez en la comunicación.

La transferencia de los datos con el esquema RLNC se hace mediante el uso de su propia cabecera. Se pueden ver dos partes diferenciadas: la primera con un tamaño de 9 bytes que corresponde con los parámetros relativos a la técnica de Network Coding, donde se incluye el tipo de paquete, *Type*, la cantidad de símbolos por generación, K, la cantidad de bits para realizar los coeficientes del vector de codificación, $GF(2^q)$, el identificador de la generación que se está transmitiendo, *Frag* y los puertos UDP correspondientes al nodo emisor y al nodo transmisor. La segunda parte de la cabecera tiene una longitud variable, que corresponde con el vector de coeficientes. Su tamaño va en función de K y del tamaño del cuerpo finito GF(2^q) y se calcula mediante la siguiente expresión: $9 + \left\lceil \frac{K \cdot q}{8} \right\rceil$ bytes. En la Figura 2.6 se muestra la disposición de la cabecera.

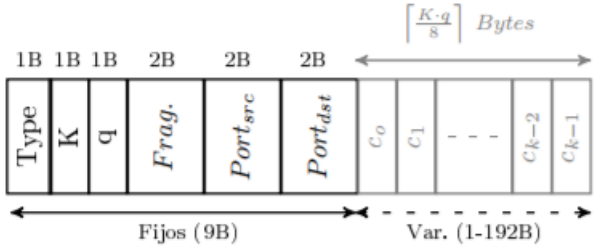


Figura 2.6: Formato de la cabecera RLNC

Para entender mejor cada campo de la cabecera se procederá a su explicación:

- ***Type***: Este campo dictamina qué tipo de paquete se está intercambiando. Puede ser una trama de confirmación (un ACK proveniente del nodo receptor) o una trama de datos.
- ***K***: Representa la cantidad de símbolos (paquetes) que contiene una generación. La elección de este parámetro puede ser crucial a la hora de determinar cuántos paquetes

descarta el receptor, ya que puede que no sean linealmente independientes a los anteriores. Esto influye en la medida del throughput.

- ***q***: El campo de este parámetro representa la cantidad de bits que se usan para determinar el tamaño del cuerpo finito GF(2^q), y en consecuencia, con la cantidad de bits que se codifican los coeficientes del vector de codificación. En este proyecto lo determinaremos a q=1 y, por tanto, solo se usaran los valores binarios 0 y 1.

- ***Frag.***: Es el número del fragmento que se está enviando, o expresándolo de otro modo, el número de generación al que pertenece el paquete que se está transmitiendo.

- $Port_{Src}$ y $Port_{Src}$: Corresponden con los puertos UDP del transmisor y el receptor. Se usan para poder distinguir los flujos de información del emisor y del receptor.

- El resto de la cabecera corresponde con los coeficientes del **vector de codificación**, c_i. Este campo es extraído por el receptor para añadirlo a la matriz C. Si el rango de la matriz no aumenta, éste será descartado, debido a que no es linealmente independiente a los anteriores. Éste campo varía en función de K y de q, su valor mínimo es cuando K=2 y q=1 (1 byte) y su valor máximo es de K=255 y q=6 (192 bytes).

2.4.2. Tunable Sparse Network Coding

El esquema *Tunable Sparse Network Coding* (TSNC) propuesto en [9] se puede ver como la evolución del esquema de NC explicado previamente en la Sección 2.4.1, RLNC. Su principal cometido era reducir el coste computacional que suponía la codificación y la decodificación en el esquema RLNC. Para esto TSNC fija un número de paquetes que se utilizan para crear un paquete codificado.

En RLNC los paquetes nativos que formaban un paquete codificado provenían de la misma generación y eran aleatorios. Sus coeficientes de codificación eran extraídos de un campo finito de Galois (GF(2^q)). En TSNC los paquetes que se utilizan para formar una generación se seleccionan a partir del conjunto K formando un subconjunto W, W=[pkt_1,pkt_2,...,pkt_w], siendo w < K. Los coeficientes de codificación, c_i, que se seleccionan son los elementos no nulos que están dentro del campo finito GF(2^q). El paquete codificado resultante se puede expresar según la Ecuación 2.2:

$$p'_{TSNC} = \sum_{i=1}^{w} c_i \cdot pkt_{j_i} \tag{2.2}$$

La elección de un número bajo de w puede suponer un incremento del rendimiento del decodificador, pero conlleva a que la probabilidad de mandar una combinación linealmente

independiente decrezca considerablemente provocando una sobrecarga en la red y perjudicando el rendimiento. Debido a la mayor probabilidad de generar paquetes linealmente dependientes a los anteriores a medida que evolucionan las transmisiones, TSNC propone la posibilidad de modificar la densidad (parámetro de NC) a lo largo de la transmisión de los datos.

A continuación se explicará la probabilidad de decodificar en el esquema RLNC. A partir de lo que se expone en [10], la probabilidad de decodificar es siguiente:

$$Prob_{r^+}^{RLNC} = 1 - \frac{Q^r}{Q^K} \tag{2.3}$$

En la ecuación 2.3 se ve que la probabilidad de recibir un paquete linealmente independiente depende del tamaño del cuerpo finito (GF) en función de r, siendo r el rango de la matriz del decodificador.

El número medio de paquetes codificados que se necesita transmitir para decodificar cuando se utiliza RLNC viene dado mediante la siguiente expresión:

$$\overline{\text{Nº de Tx}} = \sum_{i=0}^{K-1} \frac{1}{Prob_{r^+}^{RLNC}(r)} \approx k + \alpha \tag{2.4}$$

donde α es una constante que solo depende del cuerpo finito GF(Q).

Por otro lado, TSNC se modela a partir de una cadena de Markov absorbente [11], S(r,c), donde cada estado representa la probabilidad de recibir un paquete linealmente independiente al anterior, expresándolo de otra manera, aumentar el rango de la matriz que contiene el decodificador en el receptor. Los estados se representan mediante r y c. r representa el rango de la matriz que hay en el decodificador y c es la cantidad de columnas con elementos no nulos de la misma. Este modelo se define a partir de una matriz fundamental N, que depende de la probabilidad de transición entre estados.

Si tomamos el Corolario 7 de [9] se obtiene la probabilidad de recibir un paquete linealmente independiente:

$$\delta(r) = \sum_{\forall_j | (r,j) \in S} H(1,(r,j)) \cdot (1 - p_{r,j}(0,0)) \tag{2.5}$$

donde $1 - p_{r,j}(0,0)$ es la probabilidad de que el rango de la matriz no aumente y, H es la probabilidad de ir al estado j habiendo empezado en el estado de transición i. Esta depende de la matriz fundamental (N), de una matriz identidad (I) y de la inversa de una matriz diagonal, cuyos elementos no nulos coinciden con los de N, (N^{-1}):

$$H = (N - I) \cdot N^{-1} \tag{2.6}$$

A partir de las explicaciones anteriores, y la probabilidad de decodificar de RLNC se obtiene la probabilidad de decodificar de TSNC.

En la expresión 2.7 se selecciona la densidad correspondiente (p) para garantizar que la probabilidad de recibir un paquete innovador sea ligeramente menor que la que tendríamos con el esquema RLNC anterior, estando la diferencia determinada por un parámetro de configuración, $\varepsilon = 10^{-4}$. El objetivo es mantener w lo más bajo posible (para reducir la complejidad de las operaciones de NC), a la vez que se obtiene un rendimiento comparable al que se habría obtenido con RLNC en función de ε. Se disminuye w debido a que influye directamente con el rango r.

$$Prob_{r^+}^{TSNC}(r, c) \leq Prob_{r^+}^{RLNC}(r) \cdot (1 - \varepsilon) \tag{2.7}$$

La modificación de la densidad depende del valor que tome w, y que se fija para cumplir la Ecuación 2.7. Por ejemplo, el nodo origen empieza con w=1, y cada vez que se modifique el estado del decodificador se comprobará si se sigue cumpliendo la ecuación anterior. De no ser así, se aumentará w hasta que se cumpla la Ecuación 2.8.

Para que el nodo destino pueda decodificar el número máximo de paquetes se han de transmitir:

$$\overline{\text{Nº de Tx}} \leq \sum_{i=0}^{K-1} \frac{1}{Prob_{r^+}^{TSNC}(r)} \approx \frac{k + \alpha}{1 - \varepsilon} \tag{2.8}$$

2.5. UDP

El esquema de Network Coding hace uso del protocolo **User Datagram Protocol**, UDP [12] para el nivel de transporte. Este protocolo pertenece a la capa de transporte y es no orientado a la conexión, por lo tanto el intercambio de datagramas, (PDU del protocolo UDP) se realiza sin tener que establecer una conexión previa entre el transmisor y el receptor. Como característica relevante, hay que decir que no garantiza la llegada de todos los datagramas, debido a que no tiene un mecanismo de retransmisión, ni hace que los mensajes lleguen en orden, ni tiene un control de flujo. Aunque no implemente estas funciones y no sea fiable en la comunicación, la velocidad de transmisión es bastante alta.

Este protocolo es muy útil en escenarios donde se quiera transmitir información en tiempo real (videos o audio) en donde se requiere un retardo en la transmisión muy pequeño

y solo importa que los mensajes en un momento puntual más que la pérdida de paquetes.

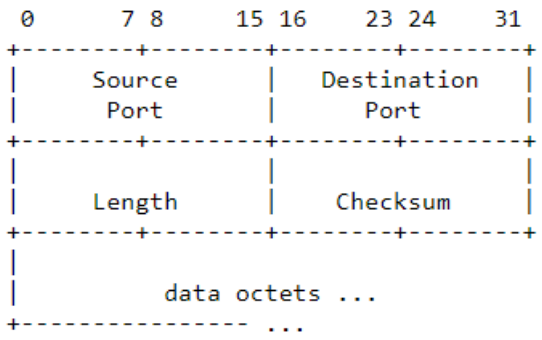


Figura 2.7: Formato de la cabecera UDP

La cabecera del protocolo UDP, Figura 2.7, se compone de los siguientes campos:

- ***Source Port*** y ***Source Port***: en este campo se especifica el puerto del nodo origen y el del nodo destino. Se componen por 16 bits cada uno y son necesarios para indicar la conexión.
- ***Length***: indica el tamaño del mensaje, la cantidad de bytes que contiene el datagrama UDP completo incluyendo la cabecera y los datos. Se compone de 2 bytes.
- ***Checksum***: este campo es un mecanismo que realiza una suma de comprobación de la combinación de una pseudocabecera IP, la cabecera UDP y los datos. Si es necesario se llevará un relleno de 0's para cuando la suma no sea múltiplo de 16 bits.
- El resto de la información que se muestra en la Figura 2.7 es el campo donde se encuentran los datos.

Debido a que este protocolo carece de ciertas funcionalidades hace que el resto de capas se tenga que hacer cargo de las deficiencias que supone. Para este trabajo no tiene relevancia estas deficiencias porque solo se usa como capa de transporte, aportando muy poca sobrecarga, debido a que en este proyecto la eficiencia de la comunicación se apoya en el esquema de NC.

2.6. Librería KODO

La liberaría KODO [13] es una librería de código abierto en lenguaje C++ que implementa los algoritmos necesarios para utilizar esquemas de codificación en red. Está diseñada para investigadores centrados en las técnicas de NC, por lo tanto, debido a que

hay distintos esquemas, a la hora de diseñarla se optó por hacerlo en pequeñas funciones personalizables por el usuario, en vez de realizar un solo bloque de un conjunto de algoritmos.

Para que la librería sea entendida por investigadores con un conocimiento limitado sobre programación, proporciona una Interfaz de Programación de Aplicaciones (API) simple, que oculta la implementación de código. Para aquellos que el lenguaje de programación no sea C++, Kodo proporciona la posibilidad de hacerlo mediante otros, como es el caso de Python.

Se pretende que la librería sea autónoma y portátil y no dependa de otras plataformas para su funcionamiento. De requerirlas, deberá proporcionar la adaptación a dicha plataforma específica. El objetivo de los desarrolladores de la librería Kodo es que los investigadores, a través de sus estudios en el área de Network Coding aporten nuevas funcionalidades que la favorezcan y así conseguir que la librería esté actualizada y en continuo progreso.

La librería Kodo es capaz de implementar varios esquemas de NC, en concreto distintos algoritmos de RLNC: en [14] se describen, aportando un conocimiento de su funcionamiento y de los entornos apropiados para usar cada algoritmo. La librería permite la aplicación del *systematic coding*. Consiste en que la fuente envía los paquetes sin codificar y, a continuación, utiliza la codificación para poder suplir las pérdidas que se hayan podido producir en el canal. Este método es muy útil en topologías simples, debido a que disminuye la sobrecarga producida por la codificación, y aumenta el rendimiento a la hora de llevar acabo la decodificación. Otra de las configuraciones que permite tener es *on-the-fly coding*, según la cual, a lo largo de la transmisión, el nodo emisor es capaz de modificar el tamaño de la generación a transmitir. Se trata de un esquema favorable en situaciones donde la cantidad de datos es variable, como podría ser el contenido en streaming. También es compatible con el mecanismo *partial decoding*, donde el receptor es capaz de decodificar parte de los datos antes de que se envíe toda la generación. Puede ser una de las grandes ventajas que ofrece la librería, debido a que a la hora de la transmisión de vídeo o audio permite que la capa de aplicación reciba los datos antes de que la generación se haya enviado completamente.

Se pueden encontrar varios esquemas RLNC como el Standard RLNC o el Sparse Network Coding con densidad uniforme o el que se utilizará en este proyecto, en el cual la densidad podrá ser modificada a lo largo de la transmisión. Por otra parte, también es compatible con diferentes métodos de codificación, como es el tradicional Reed-Solomon [15], el cual no tiene posibilidad de recodificación en los nodos intermedios y Fulcrum RLNC [16], que permite tanto a los nodos fuentes como receptores trabajar con tamaños de cuerpo superiores a $GF(2^q)$, manteniendo la compatibilidad con este.

Una de las grandes ventajas de Kodo es que permite *cross-compilation*, de esta forma, se pueden compilar ficheros en un ordenador y ejecutarlos en un dispositivo. Esta ventaja

es favorable debido a que en ocasiones los dispositivos no tiene una capacidad de cómputo elevada para realizar las operaciones pertinentes, haciendo que la compilación de esta librería sea muy pesada. En este proyecto se compila la librería en un ordenador para posteriormente ejecutarla en las Raspberry Pi.

3

Despliegue de la plataforma de experimentación

En este capítulo se expondrán las pautas que han sido seguidas para el despliegue de la plataforma de experimentación. Se relatan las herramientas que han sido necesarias para la configuración de la plataforma; desde la configuración exacta de las Raspberry Pi hasta la realización y modificación de programas para la extracción de las medidas que se explicarán en el Capítulo 4.

3.1. Materiales

Las pruebas de este proyecto se han realizado sobre un entorno real en el cual se han usado los siguientes materiales:

- 20 Raspberry Pi modelo 3B.
- 20 tarjetas SD para el almacenamiento del sistema operativo de las Raspberry Pi.
- 21 cables Ethernet.
- 20 alimentadores de corriente micro USB (>2,1 A).
- 20 pantallas.

- Un switch para conectar todas las Raspberry Pi.
- Un router para crear la red a la cual se conectarán todos los nodos.
- Una plataforma para la instalación de todo el escenario.
- Cinco regletas.
- Un módulo WiFi USB.

3.2. Desarrollo práctico

El control de la plataforma se realiza de forma remota. Toda la configuración y puesta en marcha de los experimentos se lleva a cabo mediante un ordenador y se comunica a ellos mediante la red cableada. Se definirá cada proceso y se explicará con profundidad ambos tipos de comunicaciones, la de gestión y la de experimentación.

La comunicación entre el ordenador y las Raspberry Pi se basa en el protocolo Secure Shell (SSH). Dicho protocolo sirve para la administración remota de dispositivos a través de una red y para transmitir datos de forma segura. El control remoto se puede hacer a través del terminal de Linux o en Windows usando por ejemplo xShell. xShell es un programa que permite hacer el control remoto a distintos dispositivos, y por lo tanto, resulta muy útil para este proyecto. También se puede usar el bash de Ubuntu que suministra Windows y simula el terminal de Linux. Es importante detallar que el sistema operativo que usa la Raspberry Pi es Raspbian y solo es compatible con el sistema operativo GNU/Linux, siendo Raspbian una distribución de este último.

Para realizar el control remoto hay que habilitar el puerto SSH (puerto 22) de las Raspberry Pi. En dichos dispositivos no viene por defecto, por lo que hay que modificar un fichero de configuración para habilitarlo. Cabe destacar que es importante conocer la dirección IP de la Raspberry Pi para este tipo de conexión. Lo más eficaz es utilizar un mecanismo de direccionamiento estático, siguiendo un orden a la hora de asignar las direcciones IP a cada nodo. Esta configuración se puede realizar mediante la modificación del fichero de configuración *interfaces* de este dispositivo, que se encuentra en el directorio */etc/network*. Para hacerlo de una forma menos engorrosa se optó por hacerlo a través del router, eligiendo la MAC (dirección física) correspondiente y asignando la IP más conveniente siguiendo un orden. Para acceder remotamente a este dispositivo se utiliza su usuario y contraseña, que es el método de seguridad que utilizan. Todos los dispositivos, ordenador y Raspberry Pi, tienen que estar en el mismo rango de direcciones, es decir, deben estar en la misma red.

Para hacer el control remoto a las Raspberry Pi en la red multi-salto (Sección 3.3) y que este no sea muy pesado, se ha optado por utilizar criptografía asimétrica en vez de usar

el mecanismo de usuario/contraseña. Esto consiste en que se genera una pareja de clave pública/clave privada en el ordenador copiando la clave pública en el sistema operativo de las Raspberry Pi; de manera automática se verificará la identidad del ordenador frente a los nodos de la red. Para esto se crea la pareja de claves en el ordenador mediante el siguiente comando *ssh-keygen -t rsa*. Dicha pareja se almacenará en el directorio *.ssh/* del ordenador, pudiendo acceder a él y para enviar la clave pública a las Raspberry Pi.

Host pi0
HostName 192.168.1.10
Port 22 $\longrightarrow$ Puerto del protocolo SSH
User Pi $\longrightarrow$ Usuario de la Raspberry Pi

Figura 3.1: Configuración del fichero.

Con objetivo de automatizar aún más el proceso, en el directorio mencionado anteriormente se encuentra un fichero de configuración donde se puede especificar, mediante su IP, los dispositivos donde está almacenada la clave pública y así acceder a ellos mediante un seudónimo. Por ejemplo, si se quisiera acceder a la Raspberry Pi con dirección IP 192.168.1.10 y seudónimo *pi0*, se utilizaría el fichero de configuración, que recoge la Figura 3.1.

Para este proyecto se configuró el modo de operación de la red *Ad-hoc*. Las redes inalámbricas en modo Ad-hoc son de tipo descentralizadas, debido a que no se comunican necesitando un punto de acceso. Generalmente no se conectan a otras redes más grandes, y cada nodo participa en el encaminamiento del reenvío de datos.

Para la configuración de la red en modo Ad-hoc se hizo previamente una copia de seguridad del fichero interfaces del nodo que se ha mencionado anteriormente. A continuación se añade la IP estática, se especifica la red donde está dicho nodo, el modo de operación, Ad-hoc, y el canal en el que se trabaja. La red inalámbrica a la que se accede para la configuración del panel es la que se ha creado con el router, 'RaspberryTestbed2'. Hay que tener en cuenta que todos los dispositivos que se encuentren dentro de esta red han de estar trabajando en el mismo canal.

El ordenador se conectará a la red del router, para hacer las medidas, a través de la conexión inalámbrica mediante el módulo WiFi USB.

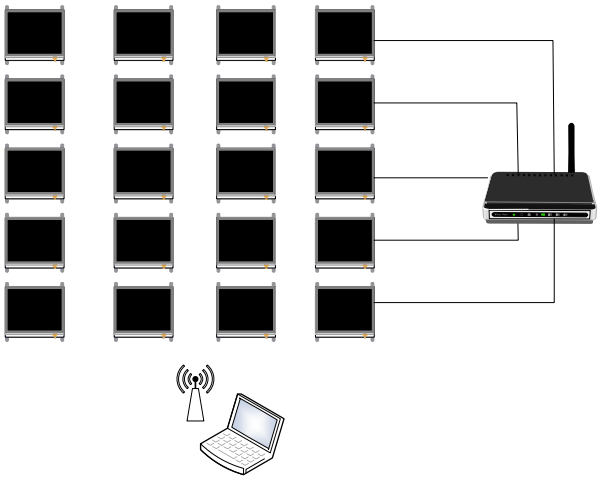


Figura 3.2: Distribución de la red

Como se ha mencionado anteriormente, todas las Raspberry Pi han de estar en la misma celda y operando en el mismo canal. Como el número de nodos de la red es elevado, se usa un switch para conectar las 20 Raspberry Pi, conectado a la red creada 'Raspberry-TestBed2' a través del router.

La estructura de la red se basará en que cuatro nodos transmiten y cuatro reciben a través de tres nodos intermedios, que puede que estén o no estén activados, conformando la red de 20 nodos, manejados remotamente con el ordenador, como se ha mencionado previamente y como se puede ver en la Figura 3.2. Los experimentos se realizan sobre la red inalámbrica, 'RaspberryTestBed2', conformada por las Raspberry Pi. Se utiliza el módulo WiFi que las propias Raspberry 3B llevan incorporado. Sobre esta red se creará la topología multi-salto en la cual se realizarán los distintos ensayos.

3.3. Desarrollo e implementación de la red multi-salto

Con la finalidad de tener una red inalámbrica multi-salto para la transferencia de archivos mediante distintas técnicas se optó por utilizar una configuración de enrutamiento estática debido a que se configura la tabla de rutas en cada dispositivo según la configuración deseada.

Para modificar la ruta o saltos de cada Raspberry Pi usaremos el comando **route**, que permite manipular la tabla de rutas de cada nodo. Este comando, como información adicional, permite especificar en el origen cual debe ser el próximo salto para llegar a un destino: **route add *destino* gw *salto* dev wlan0**.

- Destino: se pone la dirección IP del nodo al que se pretende enviar la información.
- Salto: se añade mediante la especificación 'gateway' o 'gw' y sirve para especificar el nodo intermedio por el que se encamina la información.

Para que el reenvío de la información a través del nodo intermedio se lleve a cabo hay que tener en cuenta que las Raspberry Pi tienen un fichero de configuración que hay que modificar para que funcionen como enrutador. Dicho fichero es *ip_forward*, que se encuentra en el siguiente directorio */proc/sys/net/ipv4* y hay que activarlo añadiendo un 1.

El problema de este fichero es que se resetea cada vez que se reinicia la Raspberry Pi. Adicionalmente la tabla de rutas no será siempre la misma, por lo que no todos los nodos tendrán activado el ip_forward constantemente, ni la tabla de rutas de cada uno será siempre la misma. Para automatizar estas opciones se optó por hacer un programa en Python pues facilita la implementación de las llamadas al sistema. Para realizar dicho código hay que tener en cuenta el control remoto para las especificaciones que se pretenden hacer en el sistema operativo de los nodos.

Debido a que para el acceso remoto a las Raspberry Pi hay que tener en cuenta su usuario y contraseña (como se ha mencionado en 3.2), se tendría que repetir cada vez que se ejecutara el programa. Por lo tanto se usará criptografía asimétrica, tal y como se ha explicado en la Sección 3.2.

Además de todo lo explicado, para realizar la aplicación se debe tener en cuenta que se encuentran cuatro transmisores y cuatro receptores. Por lo tanto se tienen que hacer cuatro rutas distintas y especificar las direcciones IP de cada nodo emisor y receptor. También se tienen que fijar los nodos activos entre ambos.

```
ruta[i]=[Tx[i],ON,ON,ON,Rx[i]]
Si la ruta[i] está activa:
        Establezco ruta[i]
        origen=Tx1[i]
        destino=Rx[i]
        ruta=ruta[i]
        Asig(origen, destino, ruta)
```

Figura 3.3: Programa para asignación de ruta(1)

A continuación, la función **Asig** recibe los parámetros origen, destino y la especificación de los saltos en una ruta. Dicha función habilita el enrutamiento a los nodos que se indiquen, añadiendo un 1 en el fichero ip_forward en los nodos ON y un 0 en los nodos OFF. También accede a las funciones en las cuales se determina la ruta de cada nodo enrutador (**Forward** y **Backward**).

```
Asig (origen, destino, ruta):
pi_inicial = origen - 10; ⟶ Para coincidir con el seudonimo del dispositivo.
for i = 0 : len(ruta) do
        nodo=ruta[i]
        Si nodo == ON:
                Control remoto y añadir 1 al fichero ip_forward
        Si i < len(ruta):
                Forward(destino,ruta,i) ⟶ Asignación de rutas desde origen.
        Sino i>0:
                Backward(origen,ruta,i) ⟶ Asignación de rutas desde destino.
        else:
                Control remoto y añadir 0 al fichero ip_forward
```

Figura 3.4: Programa para asignación de ruta(2)

```
Forward (destino, ruta, i):
pi_inicial = origen - 10;
for j = i+1 : len(ruta) do ⟶ j comienza en el siguiente nodo al enrutador.
        Si siguiente nodo == 1:
                Control remoto y se añade el nodo destino con el salto correspondiente.
break
```

Figura 3.5: Programa para asignación de ruta(3)

Cuando se hace las llamadas a las funciones **Forward** y **Backward** se pasan como parámetros el destino y el origen, la especificación de la ruta (los nodos habilitados) y el nodo que se está analizando en ese instante, para asignarle como nodo enrutador y añadir la nueva entrada que se incluirá en su tabla de rutas.

```
Backward (destino, ruta, i):
pi_inicial = origen - 10;
for j = nodo_enrutador : 0 do
        Si nodo_anterior==1:
                Control remoto y se añade el nodo destino con el salto correspondiente.
break
```

Figura 3.6: Programa para asignación de ruta(4)

3.4. Implementación de ficheros para Network Coding

Para el desarrollo del proyecto y la aplicación de NC hay que usar dos ficheros C++, uno que se ejecutará en los transmisores, *transmiter* y otro en los receptores, *receiver*.

Para estos dos programas se crea lo primero un socket UDP en cada uno de ellos de manera que así el transmisor y el receptor podrán comunicarse intercambiando flujo de datos.

En el emisor, *transmiter*, se genera el codificador a través de la librería Kodo, pasando como parámetros el número de símbolos por generación, el tamaño de los símbolos y el tamaño del cuerpo finito GF(Q), que en este trabajo será GF(2) (uso de un bit para realizar el vector de coeficientes propio del protocolo NC adquiriendo como valores el 1 o el 0). También se crea el buffer para almacenar los paquetes que vienen de la capa superior (capa de transporte). Debido a que el transmisor emite los paquetes codificados y recibe el ACK del receptor cada vez que le llega una generación completa, hay que incluir la función *select*, que devuelve un valor que expresa si el socket del transmisor está preparado para transmitir o recibir. Esta función se bloquea cuando el nodo está transmitiendo, y se libera a continuación, sí la tasa de transmisión al socket respeta la tasa de transmisión en el interfaz de red. Si puede transmitir lo hará mediante la función *sendto*.

En el receptor, *receiver*, después de crear el socket tal y como se ha hecho en el transmisor se usa la función *select* para saber si al socket le están llegando paquetes. Usa la función *recvfrom*, a la que le tienen que llegar paquetes para que el programa pueda ejecutarse corréctamente. Los paquetes que llegan se almacenan en un buffer. Se genera también el decodificador, que usará el nodo receptor para decodificar los símbolos que le envía el transmisor. A dicho decodificador se le pasan los mismos parámetros que al codificador. La función del receptor se limita a recibir los paquetes que se le envían, y comprueba en la cabecera correspondiente al protocolo de NC si el paquete pertenece a la generación actual, descartándolo en caso contrario. Cuando el receptor tiene todos los paquetes correspondientes a la misma generación, le manda un ACK al transmisor para que envíe la siguiente.

Para finalizar con la implementación del código se compila a través del comando de linux *make*. Para poder modificar los programas en las Raspberry Pi se usará el programa *NetBeans*, ya que permite cambiarlos a través del control remoto establecido.

Los resultados obtenidos se almacenarán en una traza en cada receptor, pero los parámetros para medir el throughput, parámetro que interesa, aparecerán por pantalla:

- Tiempo total de recepción.
- Recepciones totales.

- Recepciones válidas.
- Paquetes linealmente dependientes.
- Generaciones decodificadas.

3.5. Técnica para el banco de medidas

En este proyecto se realizan dos estudios fundamentales. La primera se basa en el estudio de las características de TCP, UDP y RLNC. En la segunda se estudiará la ventaja de TSNC sobre RLNC.

Para el primer banco de medidas se usa el programa explicado en 3.4 y la aplicación *iperf*, que es un programa cliente-servidor que permite monitorizar entornos la red. En este proyecto se utiliza para medir el throughput de los protocolos de la capa de transporte, TCP y UDP. Para cada uno de ellos se usan las siguientes especificaciones respectivamente:

Caso TCP:

- **iperf -s -i 10**: Este comando se ejecuta en la Raspberry Pi que escucha en la red, en este caso el receptor de la información. Se añade la especificación -i para que reporte medidas cada 10 segundos.

```
pi@raspberrypi:~ $ iperf -s -i 10
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[  4] local 192.168.2.20 port 5001 connected with 192.168.2.16 port 56746
[ ID] Interval       Transfer     Bandwidth
[  4]  0.0-10.0 sec  17.6 MBytes  14.8 Mbits/sec
```

Figura 3.7: *iperf* para medir el throughput en el receptor con TCP

- **iperf -c Dirección_IP_Receptor -i 10 -t 100**: Se ejecutan este comando en la Raspberry Pi que actúe como transmisor. Se pone la misma especificación para que imprima resultados cada diez segundos y se especifica el tiempo que el nodo está transmitiendo (100 segundos enviando información) para tener un número de medidas adecuado.

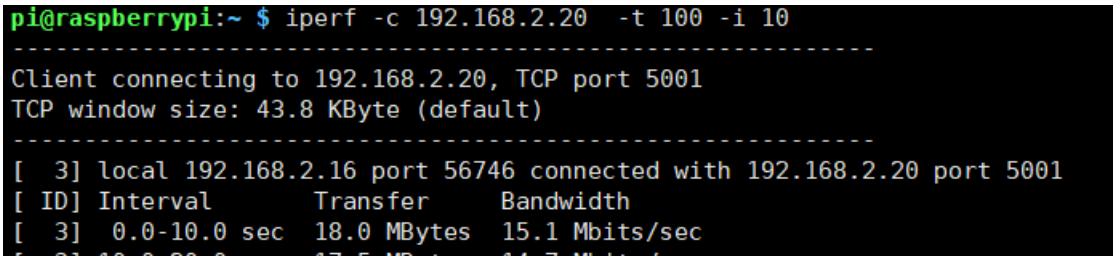


Figura 3.8: *iperf* para medir el throughput en el emisor con TCP

Caso UDP:

- **iperf -u -s -i 10**: Se ejecuta -u para el protocolo de transporte, UDP, manteniendo el resto de características.

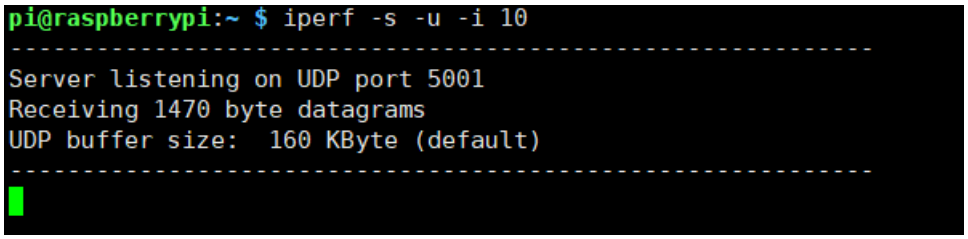


Figura 3.9: *iperf* para medir el throughput en el emisor con UDP

- **iperf -u -c Dirección_IP_Receptor -b 50M -i -t 100**: Tiene las mismas características que en el caso TCP, pero se escribe -u en el transmisor para que envíe al puerto que está escuchando en el receptor (puerto UDP) y especificar la velocidad de transmisión, 50 Mbps (-b 50M) para asegurar condiciones de saturación (en las redes Ad-hoc puede llegar a 54 Mb/s).

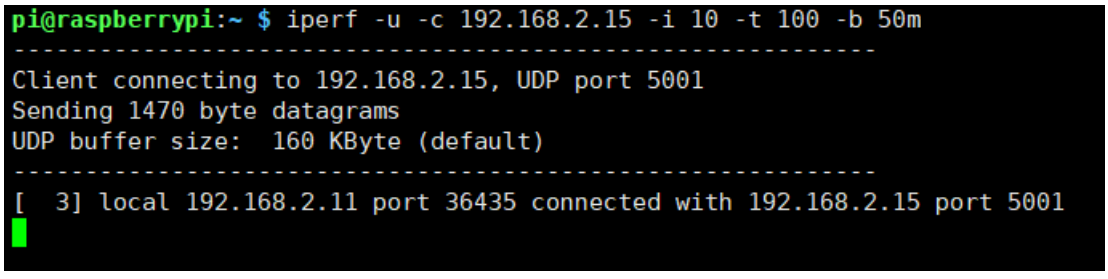


Figura 3.10: *iperf* para medir el throughput en el emisor con UDP

Para el caso del estudio de NC se varía, en el programa transmisor explicado en 3.4, el número de generaciones, el número de símbolos por generación y el tamaño de estos en bytes. Para tener unas medidas más completas se modifica el número de nodos intermedios entre el nodo transmisor y el nodo receptor, a través de la asignación de rutas utilizando la aplicación descrita en el punto 3.3.

4

Simulaciones y resultados

En este capítulo se explicarán los conocimientos necesarios para poder entender la recogida y el posterior análisis de las medidas obtenidas. A continuación, se expondrán los resultados obtenidos mediante la configuración de los parámetros de NC.

4.1. Conceptos previos

Para la recogida de medidas y su posterior estudio de ellas hay que tener presentes y los parámetros que van a ser modificados para desarrollar el experimento de una forma completa.

- **Throughput**: Se define como la cantidad de información transmitida de forma efectiva en un enlace de comunicaciones. También se puede definir como la velocidad de transferencia neta en una red, y siempre será inferior que su ancho de banda. En este trabajo se utilizará la segunda definición, calculándose mediante la Expresión 4.1. El principal objetivo de este proyecto es medir el *throughput* en una red real de los protocolos TCP y UDP, para compararles con las técnicas de NC.

$$\text{Throughput} = \frac{\text{Info. util (bytes)}}{\text{Tiempo (segundos)}} \tag{4.1}$$

- **K**: número de símbolos que contiene cada generación. Este parámetro se modificará para poder hacer varios tipos de medidas.

- **Tamaño de símbolo**: es la cantidad de bytes que contiene cada símbolo. Este parámetro se podrá modificar, pero con ciertos límites. A la hora de fijar un tamaño hay que tener en cuenta la Unidad Máxima de Transferencia (MTU), y las cabeceras de los diferentes protocolos. 8 bytes de UDP y 20 bytes de IP. En este proyecto se ha decidido fijar un tamaño de 1400 bytes.

- **Tamaño del cuerpo finito**: representado como GF(2^q). El valor a modificar en esta variable es q, que indica el tamaño que puede tener el cuerpo finito y, por lo tanto, los valores del vector de codificación han de estar dentro de él. En este proyecto se fijará a q=1, lo cual repercutirá a la cantidad de paquetes linealmente independientes. Hay que tener en cuenta la longitud del vector de codificación, debido a que variará según el número de símbolos que se transmitan y el número de bits que codifican cada elemento del vector de coeficientes, que en este caso se fijará en 1:

$$\text{Tamaño del vector codificacion} = \frac{K}{8} \tag{4.2}$$

- La red Ad-hoc se configura para trabajar a una velocidad de transmisión de 54 Mbps. Esto puede ocasionar varios problemas. Uno de los fallos se produce debido a que si la cantidad de símbolos a transmitir es muy alta, y la velocidad de transmisión también, superando la velocidad de decodificación del receptor, algunas de las generaciones no serían decodificadas, debido a que el buffer del receptor se llena demasiado rápido y algunos de los paquetes se perderían por falta de espacio en él. Si las generaciones no son decodificadas por la pérdida de paquetes, con el consecuente descenso del rendimiento de la red. Debido a esto las redes WiFi tiene un mecanismo (*Adaptative Rate Selection*), que consiste en adaptar la velocidad de transmisión según los paquetes perdidos. Si la pérdida es muy alta la velocidad de transmisión puede reducirse los 2 Mbps.

- Adicionalmente, se ha de tener en cuenta la velocidad de decodificación que es capaz de ofrecer la Raspberry Pi 3 modelo B, ya que influye directamente al throughput obtenido en las medidas realizadas.

- **Paquetes fuera de generación**: estos paquetes son aquellos que se reciben y pertenecen a una generación ya recibida y confirmada. Esta cantidad debería ser muy pequeña, pero debido a que no se tiene ningún control sobre el buffer del WiFi para borrar las tramas que contienen los paquetes pertenecientes a otra generación ya confirmada, no puede obviarse.

- **Retransmisiones a nivel 802.11**: El estándar 802.11 [17] especifica las capas de acceso al medio (capa MAC y física) de las redes inalámbricas. Proporciona un número finito de retransmisiones cada vez que un paquete no llega al nodo destino. Esta

característica puede alterar el rendimiento de la red, y especialmente cuando se hace uso del protocolo UDP en modo unicast. Dichas retransmisiones no se pueden quitar u omitir, debido al escaso control sobre las tarjetas. Si se trabajase en modo broadcast o multicast, el nivel 802.11 no haría uso de las retransmisiones, y TCP se vería notablemente perjudicado.

- **Frame Error Rate (FER)**: Este parámetro indica el porcentaje de pérdidas de las tramas en el canal. Será un valor a considerar en el estudio de los resultados.

4.2. Descripción del escenario y proceso de medida

El escenario que se usará para la extracción de medidas es el representado en la figura 3.2. Los nodos superiores serán los transmisores y los nodos inferiores los receptores. Las Raspberry Pi intermedias se usan para direccionar los paquetes desde el origen hasta el destino, produciendo una comunicación unicast.

Para el proceso de medidas, las herramientas utilizadas son las que se han explicado en el punto 3.5. Se usará el programa *iperf* para medir el throughput en los protocolos UDP y TCP para compararlo con el uso de las técnicas de Network Coding basadas en el protocolo UDP como capa de transporte. Para el estudio de NC, concretamente RLNC, se usará el programa explicado en la sección 3.4 tanto en el transmisor como en el receptor, pero en modo *best effort*. Los nodos intermedios se configurarán para que cada una de las cuatro rutas posibles tenga un número de saltos distinto. Para que las medidas sean favorables se probará cada ruta sin activar las demás, debido a que se comparte el canal y el resultado del throughput no sería el más correcto.

A continuación se llevará a cabo una comparación entre los resultados experimentales del esquema RLNC y los valores teóricos obtenidos mediante el esquema TSNC.

Para poder analizar de forma objetiva los resultados se cogerán 30 medidas del throughput en cada una de las rutas configuradas con distintos números de saltos usando los protocolos TCP y UDP para calcular el valor medio de todos los resultados del rendimiento, $\overline{throughput}$. Para las medidas realizadas bajo el esquema NC se harán 50 realizaciones para cada valor de K asignado. En los resultados obtenidos mediante la técnica de NC se obtiene el tiempo total requerido para la recepción de los datos, las generaciones decodificadas, los paquetes que se han transmitido y los paquetes que han accedido al decodificador para poder decodificar dichas generaciones.

4.3. Resultados

En esta sección se analizan medidas resultantes de los protocolos UDP, TCP y las técnicas de NC para su posterior comparación. Se deben tener presentes los conceptos explicados en la sección 4.1.

4.3.1. Análisis de los resultados

El principal objetivo en este apartado es comprobar el thoughput que ofrecen los protocolos de la capa de transporte TCP y UDP frente a las técnicas de NC.

Para la obtención de las medidas de TCP se usará el programa *iperf*. Para hacer la medida en la ruta que se seleccione hay que poner al nodo receptor escuchando en el puerto TCP (Figura 3.7) y el nodo transmisor a generar paquetes (Figura 3.8). Como el escenario se compone por 4 rutas posibles, a cada ruta se le asigna un número de saltos entre origen y destino, para ver como el thoughput varía en función del número de saltos.

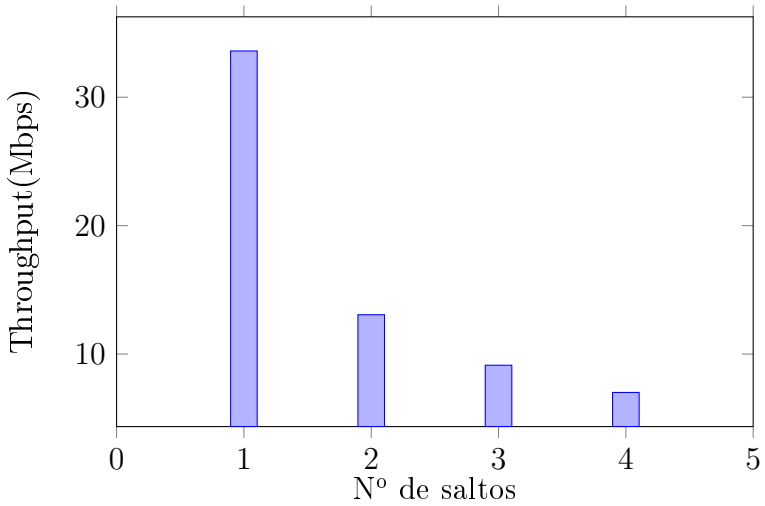


Figura 4.1: Medida del throughput con TCP

En la Figura 4.1 se muestra el $\overline{throughput}$ resultante mediante TCP. Se puede ver que este disminuye a medida que se introducen nuevos saltos entre el transmisor y el receptor. Si la comunicación es directa entre el origen y el destino (un salto) se puede obtener un $\overline{throughput}$ de 33.6 Mbps, mientras que introduciendo cuatro saltos, disminuye a 7.01 Mbps.

Para el estudio del protocolo UDP se siguen los mismos pasos que con TCP, usando el programa *iperf* (Figuras 3.9 y 3.10) para el análisis del rendimiento de la red.

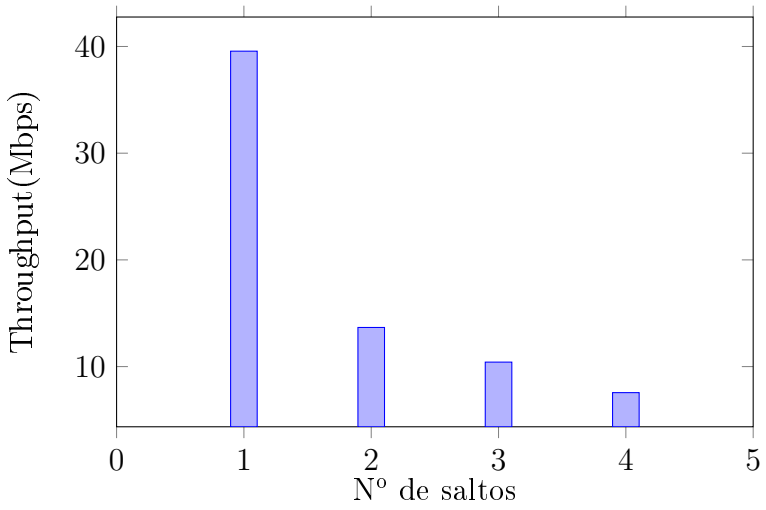


Figura 4.2: Medida del throughput con UDP

En la Figura 4.2 se muestra el $\overline{throughput}$ bajo las mismas condiciones que TCP. Teóricamente, el rendimiento mediante UDP disminuye a medida que se van aumentando el número de saltos entre el origen y el destino: con S saltos el rendimiento de la comunicación directa (con un salto) se divide entre S ($\frac{1}{S}$). Se puede comprobar que este protocolo se comporta mejor, debido a que no genera reconocimientos, con un rendimiento medio de 39.56 Mbps. A medida que se aumentan los saltos intermedios, el $\overline{throughput}$ de UDP disminuye, manteniéndose siempre por encima de TCP.

Para el análisis de medidas del esquema RLNC se ha utilizado el programa descrito en la Sección 3.4, basándose en la librería Kodo. Se ha utilizado, como se ha mencionado anteriormente, el modo *best effort*, de tal manera que cada vez que se recibe una generación el nodo receptor no manda su confirmación. Para la caracterización del $\overline{throughput}$ de esta técnica se hará uso de distintos valores del parámetro K (64, 128, 255), la cantidad de bytes por símbolo (L) y el número de generaciones (N):

$$\overline{Throughput} = \frac{\text{Info. util}}{\text{Tiempo}} = \frac{K \cdot L \cdot N \cdot 8 \text{ (bytes)}}{\text{Tiempo (segundos)}} \tag{4.3}$$

En la siguiente Tabla 4.1 se muestran los valores que se han utilizado para obtener el rendimiento de la red independientemente del número de saltos:

Tabla 4.1: Parámetros para el cálculo del $\overline{Throughput}$

Número de símbolos (K)	Tamaño del símbolo en bytes (L)	Nº de generaciones (N)
64	1400	100
128	1400	100
255	1400	100

El resultado del $\overline{throughput}$ en función del número de saltos y el parámetro K es el siguiente:

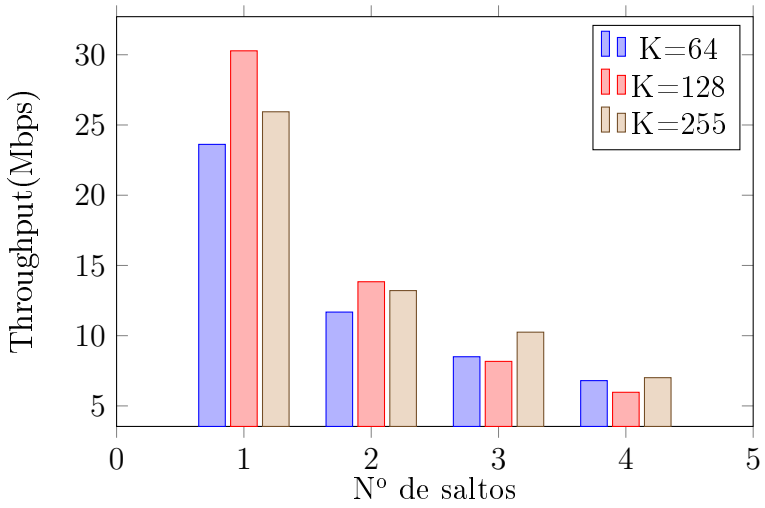


Figura 4.3: Medida del $\overline{throughput}$ con RLNC

Examinando la Figura 4.4 se comprueba que el rendimiento utilizando la técnica RLNC varía pudiendo llegar a un valor máximo de 30,28 Mbps en el caso de que la comunicación fuera directa y con un valor de K = 128. En los estudios previos (simulación) este valor es notablemente mayor que el obtenido por el protocolo TCP, pero sin superar al rendimiento dado por UDP. En el caso de un entorno real, el resultado no es el esperado, y el rendimiento con este esquema de NC no permite llegar a un throughput que pueda superar el que se obtiene con TCP, debido a la limitación de la velocidad del decodificador en el receptor, parámetro que no se tenía en cuenta en el entorno de simulación. Por otro lado, cuando K = 255 se consigue llegar a 25,94 Mbps, superando el de las simulaciones, pero sin la suficiente fiabilidad respecto a la información transmitida.

Se puede observar que a medida que se aumentan los saltos entre el transmisor y el receptor, el rendimiento disminuye. A esto, si se le suma el incremento de la K el $\overline{throughput}$ es menor, debido a que la cantidad de paquetes a decodificar es mayor, incrementando así el rango de la matriz en el decodificador y, en consecuencia, el grado de complejidad en las operaciones que debe realizar.

El problema que se observa a la hora de analizar las medidas es que el $\overline{throughput}$ de UDP, teóricamente, debería quedar notablemente por encima que el de TCP:

Tabla 4.2: Resultados en el área de simulación del $\overline{throughput}$ (Mbps)

Protocolo	1 salto	2 saltos	3 saltos	4 saltos
TCP	19.6	10.28	7.17	5.47
UDP	23.27	12.66	8.67	6.56
NC	22.81	11.77	7.83	5.56

Como se observa en la Tabla 4.2 el rendimiento de NC queda por encima del protocolo TCP, independientemente de los saltos que haya entre origen y destino con una K = 128. La Figura 4.4 representa los resultados prácticos cuando K = 128, respecto a los otros dos protocolos:

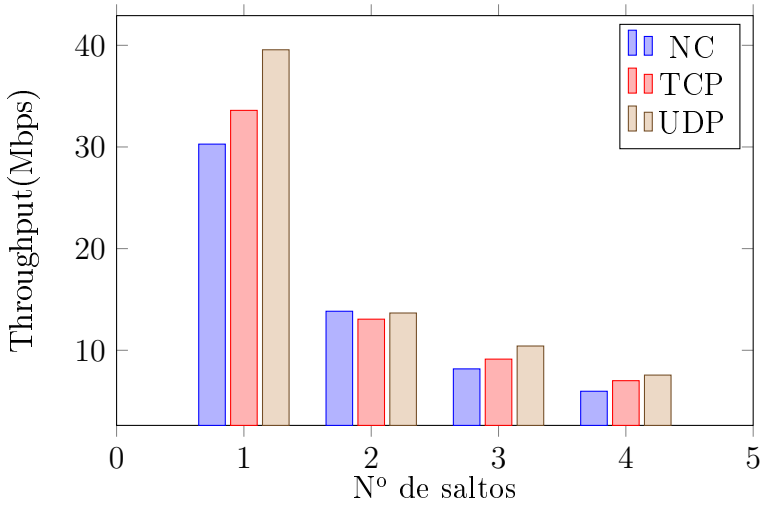


Figura 4.4: Medida del $\overline{throughput}$ en el entorno real

se observa que el $\overline{throughput}$ de TCP es ligeramente superior al de NC, y simular al de UDP. Este resultado se debe a las retransmisiones que se producen a nivel 802.11. Dichas retransmisiones benefician al protocolo TCP ya que, como es bien sabido, este protocolo no es el idóneo para las adversidades que se producen en el medio radio, porque no sabe distinguir si los problemas son por congestión o por pérdidas, por lo tanto el estándar 802.11 ayuda a recuperar la información que se pierda favoreciendo a la ventana de congestión de TCP. Por otro lado, a UDP le puede perjudicar, debido al tiempo invertido en el proceso de retransmisión 802.11.

Tabla 4.3: Simulación del $\overline{throughput}$ sin retransmisiones 802.11 (Mbps)

Protocolo	1 salto	2 saltos	3 saltos	4 saltos
TCP	3.26	0.9	0.5	0.26
UDP	23.21	12.9	8	5.76

En la Tabla 4.3 se ve el resultado de las simulaciones en el caso de que no existieran las retransmisiones a nivel 802.11. Se puede ver como el rendimiento de TCP decae notablemente cuando no existen las retransmisiones, y UDP apenas se ve perjudicado. En el caso de este trabajo, las retransmisiones no se pueden quitar, debido a las limitaciones del driver de la interfaz inalámbrica de las Raspberry Pi. Si la comunicación fuera en modo broadcast o multicast las retransmisiones a nivel 802.11 no se aplican y por lo tanto el comportamiento de TCP se vería perjudicado, porque la pérdida de paquetes sería superior.

4.4. Mejora sobre RLNC

Otro aspecto interesante a analizar es la fiabilidad en la comunicación mediante la probabilidad de decodificar. Como se ha explicado en la Sección 2.4.2, esta probabilidad está diréctamente relacionada con la densidad (p). p viene a ser la probabilidad de que haya elementos nulos en el vector de codificación. Es importante tener en cuenta este valor debido a que a medida que aumentan las recepciones y la cantidad de elementos no nulos es pequeño, la probabilidad de que llegue un paquete linealmente independiente decae. En el esquema RLNC este parámetro, por defecto, corresponde a 0.5.

En la Figura 4.5 se muestra la influencia en el cambio de p, ($p = 100$), y cuando K=100. Si p aumenta la cantidad de paquetes adicionales que el receptor que necesita para decodificar aumenta, es decir, la redundancia de los K paquetes es mayor, necesitando en el receptor $N = K + \beta$ paquetes para que la probabilidad de decodificar aumente, siendo β los paquetes de más que necesita el decodificador. Esto conduciría a un aumento en la complejidad de las operaciones en el decodificador, reduciendo el rendimiento y aumentando el consumo energético.

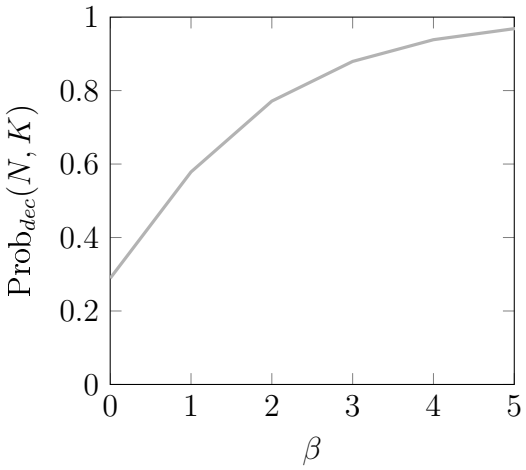


Figura 4.5: Sobrecarga RLNC, $k = 100$

Debido a que la densidad en el esquema RLNC es constante y no se puede modificar, se propuso el esquema TSNC. Según la respuesta del decodificador, modificará la cantidad de paquetes w, tal y como se explica en 2.4.2. La densidad está relacionada diréctamente con w debido a que si p es la probabilidad de que haya términos nulos en el vector de codificación, w es la cantidad de elementos no nulos que hay en dicho vector. Por lo tanto, TSNC modifica la cantidad de paquetes a transmitir mediante el incremento de p, $w = (1 - p) \cdot K$.

El cambio de w mediante la técnica TSNC hace que el throughput del decodificador aumente, ofreciendo así un gasto de energía menor. En la Figura 4.6 se compara el comportamiento del decodificador en las Raspberry Pi modelos 2 y 3, con el esquema RLNC ($p = 0.5$) y con el esquema TSNC ($p \neq 0.5$).

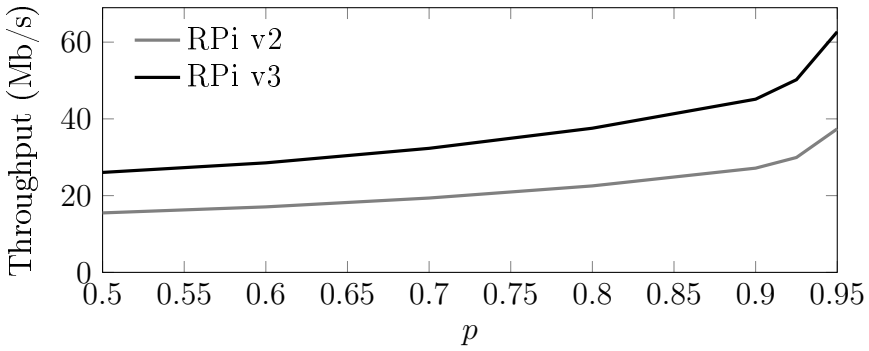


Figura 4.6: Throughput del decodificador sobre diferentes dispositivos (RPi v2 and RPi v3) para diferentes valores de densidad, p

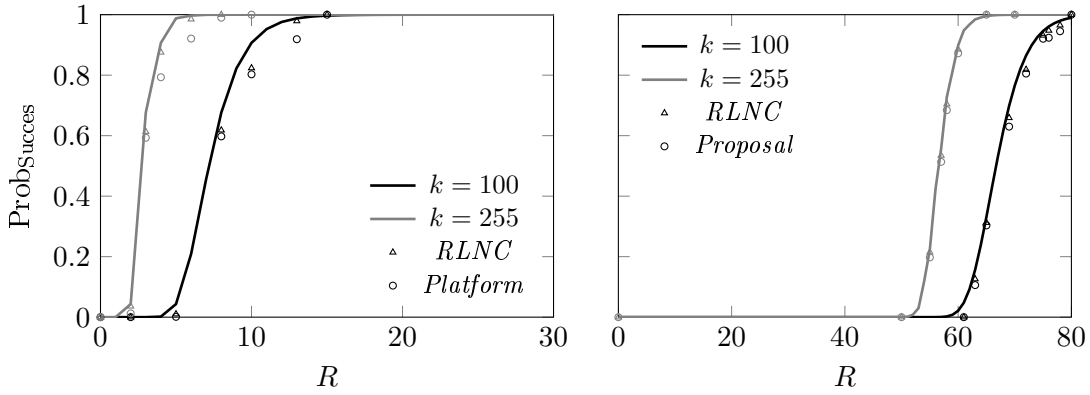
Se puede comprobar que el esquema TSNC aporta un mayor rendimiento del deco-

dificador debido a que la cantidad de paquetes que se necesitan para decodificar es menor. Por lo tanto, la complejidad de decodificar disminuye debido a la cantidad de operaciones que usa para la correcta decodificación de las generaciones. En simulaciones se ha conseguido calcular la cantidad de operaciones necesarias para ambas técnicas con la misma transmisión de paquetes.

Tabla 4.4: Resultados en el área de simulación del número de operaciones para los esquemas RLNC y TSNC.

Esquema	Throughput (Mbps)	Operaciones	Transmisiones
RLNC	26	9953	4100
TSNC	91,24	2091	4100

Para que se puedan asentar los conceptos, en la Figura 4.7 se representan los valores téoricos utilizando la Ecuación 2.7 para la probabilidad de éxito, modificando la redundancia (β). Se comparan las técnicas RLNC y TSNC respecto a la probabilidad de éxito comprobando que se equiparan, de forma que el throughput de la red se mantiene igual en ambos esquemas, modificando símplemente el estado del decodificador. Se puede ver que a medida que se aumenta la redundancia, la probabilidad de recibir toda la información correctamente aumenta.



(a) $FER = 0.1$ (b) $FER = 0.3$

Figura 4.7: Probabilidad de éxito en la racepción de información con distintos valores de redundancia, ρ. Las líneas sólidas corresponden con los valores teórricos y los marcadores representan los valores experimentales.

5

Conclusiones y líneas futuras

En este capítulo se completa la memoria, exponiendo las conclusiones a las que se han llegado y un resumen de la comparación de los protocolos utilizados. En la sección 5.2 se explican las líneas del trabajo que se han abierto respecto al estudio de las técnicas de NC en entornos reales, y su posible implementación para la transmisión de datos.

5.1. Conclusiones

Debido principalmente al auge de las comunicaciones inalámbricas y los problemas a los que éstas se enfrentan, originados por unas condiciones adversas del canal radio sobre el que se llevan a cabo, se tiende a estudiar las posibles formas de conseguir que no influyan en las comunicaciones. Uno de los principales campos de investigación en esa línea son las técnicas de NC, contando ya con numerosos resultados a través de entornos simulados. Debido a que hasta el momento no se han realizado demasiadas pruebas en entornos reales, este proyecto se centra en la construcción de un escenario de experimentación real para estudiar el comportamiento de esta técnica sobre redes multi-salto.

Los esquemas que se están estudiando en el grupo de investigación son RLNC y TSNC. Este proyecto se centrará en la creación de un entorno real de trabajo para aplicar la técnica RLNC. Si bien es cierto que ya se han obtenido numerosos resultados en el ámbito de la simulación y se hizo una pequeña introducción para la implementación en un minicomputador, este trabajo pretende ser el punto de partida para la caracterización del

comportamiento de los esquemas de codificación en red. Del estudio llevado a cabo durante el proyecto se pueden extraer varias conclusiones, tanto en el aspecto instrumental como en el experimental.

Por un lado, se estableció que el entorno de trabajo debía ser una red multi-salto con el modo de operación *Ad-hoc*, a pesar de que hasta el momento solo se había experimentado en modo infraestructura. La configuración de la red se pudo implementar consiguiendo su perfecto funcionamiento, concretamente creando la red multi-salto.

Por un lado, el esquema RLNC se ha analizado según las diferentes respuestas que ha dado a partir de la modificación de su configuración. En primer lugar, se ha variado la cantidad de símbolos que contiene una generación (K), para comprobar el comportamiento del algoritmo. Los valores que se han escogido son 64, 128 y 255. Según los resultados obtenidos con anterioridad referentes a las respuestas frente a dicha modificación, los observados en la red multi-salto difieren respecto a lo esperado. La configuración de K=255 no es la mejor respecto al throughput de la red, pero su rendimiento es bastante superior a lo conseguido en los entornos de simulación. Al configurar el número de saltos que hay en cada ruta, se ha comprobado que el rendimiento disminuye a medida que se aumenta el número de saltos.

El último resultado obtenido, es el referente a su comparativa con el rendimiento de TCP. Se esperaba que el rendimiento resultante por el esquema RLNC superara al de TCP sin llegar al ofrecido por UDP, pero en el análisis de la plataforma se comprobó que no es así. A medida que los saltos aumentaban entre el origen y el destino el, throughput entre TCP y UDP se iban asemejando aunque este último fuera superior. La sorprendente respuesta de TCP puede venir dado a que el sistema operativo de las Raspberry Pi incorpora una versión de este protocolo bastante mejorada (Cubic) y a la respuesta que ofrece respecto a las retransmisiones de los paquetes que se hayan podido perder. Otra de las razones del dominio de TCP frente a UDP es que las retransmisiones a nivel 802.11 favorecen en gran medida al protocolo TCP, como se ha explicado anterioremente. Esta deficiencia influye al esquema RLNC, debido a que no se necesitan retransmisiones porque la información que se manda entre dos nodos es siempre la misma, es decir, se hace una combinación lineal de la misma información, por lo que un paquete perdido no afecta a la fiabilidad de la comunicación, ya que puede ser sustituido por otro. Hasta el momento, en las Raspberry Pi, no es posible omitir las retransmisiones de 802.11.

En conclusión, el problema principal que se encuentra es en referencia a nivel 802.11. Se puede decir que las técnicas de NC en redes multi-salto cuando se trabaja en modo unicast, se ven perjudicadas debido a las retransmisiones que se producen en la capa de acceso al medio, lo que beneficia al protocolo TCP y mejorando su rendimiento sobre estas redes. Si se configurase la red en modo broadcast/multicast, el nivel 802.11 no usaría las retransmisiones perjudicando al protocolo TCP y pudiendo favorecer este hecho a las técnicas de NC.

Con este proyecto se pretende dejar claro los problemas que supone llevar a un entorno real las técnicas de Network Coding.

5.2. Líneas futuras

En esta sección se introducen algunas de las posibles lineas futuras que quedan abiertas para continuar con el trabajo realizado en este proyecto.

A pesar de que las técnicas de Network Coding son bastante recientes y se encuentran en desarrollo, y que este trabajo se ha enfocado más en la construcción de un entorno real como es el despliegue de la plataforma para poner en práctica NC, viendo los resultados obtenidos, una línea de estudio sería la posibilidad de implementar este esquema en redes más complejas, como serían aquellas que estuvieran en modo broadcast/multicast, y ver el comportamiento que se obtiene respecto a soluciones más tradicionales en dicho escenario.

Otro punto de vista sería seguir con este esquema y las características del escenario, para poder profundizar en los resultados y caracterizar de forma definitiva los problemas que se encuentran. Como se ha dicho anteriormente, el problema principal son las retransmisiones a nivel 802.11 cuando el modo de operación es unicast, por lo tanto se podría estudiar la manera de omitir o ignorar dichas retransmisiones, para que al esquema RLNC no le afecten y así obtener un throughput favorable y una comunicación fiable cuando se usa codificación en red.

Por otro lado, también se puede usar las técnicas de NC y Multi-path en un entorno real como es la plataforma que se ha realizado. Para esto el modelo de Raspberry es ideal debido a que tienen el módulo WiFi integrado además de puertos USB donde se podría conectar otra interfaz para realizar la técnica Multi-path. En [18] se explica en profundidad la combinación de ambas técnicas.

Siguiendo con la línea de este proyecto, con la red multi-salto, se podría implementar la posibilidad de recodificar en los nodos intermedios y debido a que en este trabajo solo redirigen la información hacia el nodo destino.

Por último, se podría estudiar el comportamiento al transmitir vídeo mediante estas técnicas, sobre las topologías de red que se han analizado.

Bibliografía

[1] Ramón Agüero Calvo. *Contribución a la mejora de las prestaciones en redes de acceso inalámbricas no convencionales*. PhD thesis, Universidad de Cantabria, 2008.

[2] Rudolf Ahlswede, Ning Cai, S-YR Li, and Raymond W Yeung. Network information flow. *IEEE Transactions on information theory*, 46(4):1204–1216, 2000.

[3] David Gómez, Eduardo Rodríguez, Ramón Agüero, and Luis Muñoz. Reliable communications over wireless mesh networks with inter and intra-flow network coding. In *Proceedings of the 2014 Workshop on ns-3*, page 4. ACM, 2014.

[4] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. *IEEE/ACM Transactions on Networking (ToN)*, 16(3):497–510, 2008.

[5] Eduardo Rodríguez Maza et al. Combinación de técnicas de network coding y codificación lineal aleatoria sobre redes malladas inalámbricas. 2014.

[6] Szymon Chachulski, Michael Jennings, Sachin Katti, and Dina Katabi. *Trading structure for randomness in wireless opportunistic routing*, volume 37. ACM, 2007.

[7] Tracey Ho, Ralf Koetter, Muriel Medard, David R Karger, and Michelle Effros. The benefits of coding over routing in a randomized setting. 2003.

[8] David Gómez, Eduardo Rodríguez, Ramón Agüero, and Luis Muñoz. Reliable communications over lossy wireless channels by means of the combination of udp and random linear coding. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–6. IEEE, 2014.

[9] Pablo Garrido, Daniel E Lucani, and Ramón Agüero. Markov chain model for the decoding probability of sparse network coding. *IEEE Transactions on Communications*, 65(4):1675–1685, 2017.

[10] Oscar Trullols-Cruces, Jose M Barcelo-Ordinas, and Marco Fiore. Exact decoding probability under random linear network coding. *IEEE communications letters*, 15(1):67–69, 2011.

[11] Pablo Garrido, Daniel E Lucani, and Ramon Aguero. How to tune sparse network coding over wireless links. In *Wireless Communications and Networking Conference (WCNC), 2017 IEEE*, pages 1–6. IEEE, 2017.

[12] Lars-Ake Larzon, Mikael Degermark, Stephen Pink, Lars-Erik Jonsson, and Godred Fairhurst. The lightweight user datagram protocol (udp-lite). Technical report, 2004.

[13] Morten V Pedersen, Janus Heide, and Frank HP Fitzek. Kodo: An open and research oriented network coding library. In *International Conference on Research in Networking*, pages 145–152. Springer, 2011.

[14] Librería kodo. http://docs.steinwurf.com/overview.html.

[15] Jérôme Lacan, Vincent Roca, Jani Peltotalo, and Sami Peltotalo. Reed-solomon forward error correction (fec) schemes. Technical report, 2009.

[16] Daniel E Lucani, Morten V Pedersen, Diego Ruano, Chres W Sørensen, Frank HP Fitzek, Janus Heide, and Olav Geil. Fulcrum network codes: A code for fluid allocation of complexity. *arXiv preprint arXiv:1404.6620*, 2014.

[17] Neil Reid and Ron Seide. *Manual de Redes Inalámbricas 802.11 (Wi-Fi)*. McGraw-Hill, 2005.

[18] David Gómez Fernández et al. Uso de técnicas de network coding y multipath para la mejora de las comunicaciones sobre redes malladas inalámbricas. 2015.

[19] Raquel Fernández González et al. Implementación de técnicas de codificación sobre una plataforma raspberry-pi. 2016.

Lista de acrónimos

WMN Wireless Mesh Network

NC Network Coding

TCP Transmission Control Protocol

UDP User Data Protocol

IP Internet Protocol

COPE Opportunistic Coding in Practical Wireless Network Environment

MORE MAC-independent opportunistic routing protocol

RLNC Random Linear Network Coding

ACK Acknowledgement

TSNC Tuneble Sparse Network Coding

API Interfaz de Programación de Aplicaciones

SSH Secure Shell

MTU Unidad Máxima de Transferencia

FER Frame Error Rate