ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

UNIVERSIDAD DE CANTABRIA

Trabajo Fin de Grado

Análisis de los beneficios de la recodificación en soluciones RLNC: caracterización sobre una plataforma basada en Raspberry Pi

On the benefits of recoding within RLNC solutions: characterization over a Raspberry Pi based platform

Para acceder al Titulo de

Graduado en Ingeniería de Tecnologías de Telecomunicación

Autor: Alfonso Fernández Gutiérrez

Junio - 2018

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

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

CALIFICACIÓN DEL TRABAJO FIN DE GRADO

Realizado por: Alfonso Fernández Gutiérrez
Director del TFG: Ramón Agüero Calvo y Pablo Garrido Ortiz
Título: "Análisis de los beneficios de la recodificación en soluciones RLNC: caracterización sobre una plataforma basada en Raspberry Pi"
Title: "On the benefits of recoding within RLNC solutions: characterization over a Raspberry Pi based platform"

Presentado a examen el día: 23/06/2018

para acceder al Título de

GRADUADO EN INGENIERÍA DE TECNOLOGÍAS DE TELECOMUNICACIÓN

Composición del Tribunal:
Presidente (Apellidos, Nombre): García Arranz, Marta
Secretario (Apellidos, Nombre): Agüero Calvo, Ramón
Vocal (Apellidos, Nombre): Fernandez Solorzano, Victor

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

Resumen ejecutivo

Es una realidad que hoy en día tanto las redes como los dispositivos conectados a ellas están cambiando. Es fácil apreciar como las velocidades exigidas por las aplicaciones y los usuarios son cada vez mayores. Dentro de estas nuevas redes cobran especial atención las redes inalámbricas, con un crecimiento exponencial en el número de usuarios, podemos destacar áreas emergentes de las telecomunicaciones como puede ser Internet of the Things (IoT) o las futuras redes móviles 5G.

Debido a los inconvenientes que aparecen en el entorno inalámbrico, se están investigando diferentes técnicas capaces de satisfacer estas demandas. Entre ellas cabe destacar el *Network Coding*. Dentro de esta técnica hay diferentes variantes, este trabajo se va a centrar en Random Linear Network Coding (RLNC).

En este documento se analizan dichas técnicas sobre escenarios reales. Concretamente las medidas que han sido realizadas mediante un entorno multi-salto con diferentes variantes sobre las tareas a realizar por los nodos intermedios. Estos resultados han sido obtenidos mediante dos vías: La primera de ella a través de simulación utilizando el software NS-3, la segunda forma ha sido mediante el uso de un banco de pruebas compuesto por Raspberry Pi 3.

Abstract

It is reality that nowadays networks as well as devices connected to them are changing. It is easy to see that the required connectivity speed by the applications and users is growing faster each day. Inside those emerging networks wireless ones have great significance, growing exponentially the number of users. We can stand out some telecommunication areas like IoT or the next mobile generation networks (5G).

Due to the inconveniences that merge in the wireless environment, different technics capable of satisfying the demand are being researched. One of these techniques is *Network Coding*. Within this technique there are many variants, however, this project is focused on RLNC.

In this document many results from possible real scenarios appear; specifically, the statics that have been measured into a multi-leap with several variants over the intermediate nodes' tasks. These results have been obtained by two methods: The first one by software NS-3, the other by a benchmark composed by Raspberry Pi 3.

Agradecimientos

En primer lugar, quiero agradecer a Pablo y a Ramón la ayuda que me han prestado ya que sin ellos no hubiera sido capaz de sacar adelante este trabajo, siempre que he necesitado consejo han estado dispuestos a ayudarme.

También debo agradecer a mi familia especialmente a mis padres todo el apoyo que me han mostrado, no solo para la elaboración de este trabajo sino a lo largo de todo el Grado.

Por último, dar gracias a mis amigos y compañeros de clase con los cuales he podido alejarme de las preocupaciones de los estudios siempre que lo he necesitado.

1 Introducción

En la primera parte de este documento se plantea el problema que motiva la realización de este trabajo así como los resultados que se pretenden obtener. Por último, aparece un breve resumen de la estructura del documento y de las diferentes partes que lo componen.

1.1. Planteamiento del problema

Desde hace unos años venimos experimentando una gran revolución en lo que se refiere al ámbito de las telecomunicaciones. Esto está especialmente marcado por las redes inalámbricas, que cada día experimentan un mayor crecimiento. También cabe destacar los nuevos servicios que se ofrecen como: Video on Demand (VoD), Streaming, ... los cuales requieren cada vez mayores prestaciones.

El principal problema que aparece en las redes inalámbricas es la baja calidad del enlace que presentan, debido a las interferencias, ruido, desvanecimientos...

El protocolo tradicional que se usa para asegurar la recepción de la información extremo a extremo es Transport Control Protocol (TCP) es un protocolo que fue diseñado para entornos cableados. El uso de TCP sobre redes inalámbricas provoca una gran caída en términos de eficiencia, debido al control de la de la congestión, el cual no es capaz de distinguir entre pérdidas debido a la congestión o provocadas por interferencias. Como resultado,

el rendimiento de TCP sobre las redes inalámbricas se ve fuertemente perjudicado.

En este trabajo se propone utilizar UDP, el cual no asegura la recepción de la información, junto con *Network Coding* (NC),el cual esta garantizando la recepción de los datos. Estudiaremos esta propuesta sobre topologías multi-salto donde los nodos intermedios tienen capacidad de combinar la información.

1.2. Objetivos

Los principales objetivos que se quieren alcanzar con este trabajo son:

- Desarrollo de una plataforma de experimentación.
- Evaluación de las técnicas de NC sobre dicha plataforma.
- Corroborar y complementar los resultados en un entorno de simulación.

Hoy en día **NC** es una de las técnicas más estudiadas en el entorno inalámbricos, ya que NC rompe con la filosofía de transmisión clásica, ofreciendo un mecanismo de transmisión eficiente sobre entornos inalámbricos. Sin embargo, pocos trabajos han mostrado las ventajas de NC en implementaciones reales. Además, el hecho de que los nodos intermedios contribuyan a la robustez de la transmisión, a través de la recodificación, es un aspecto bastante novedoso del cual no existen muchos resultados.

Este trabajo se centra en tratar de caracterizar las prestaciones de NC, comparando los resultados con esquemas tradicionales de transmisión. Se estudia el uso de recodificación en los nodos intermedios. Se describe que escenarios son los indicados para usar esta técnica, ya que el uso de NC de forma óptima es dependiente de las características del medio.

Las medidas han sido obtenidas en un banco de pruebas real compuesto por 20 Raspberry Pi 3. Por otra parte, algunos de los resultados de esta plataforma han sido contrastados con los obtenidos mediante la simulación en el software NS-3.

Tanto para el entorno de simulación como en el despliegue sobre dispositivos reales el lenguaje de programación ha sido C++.

1.3. Estructura de la memoria

Ahora se realiza un pequeño resumen de la estructura del documento, explicando brevemente los contenidos de cada uno de los capítulos.

- Capítulo 2: Se presentarán los conceptos teóricos que han dado pie a plantear este trabajo. Se explica de forma detallada todos los conceptos necesarios para el desarrollo y entendimiento del trabajo. Se estudian los protocolos utilizados a día de hoy para la transmisión sobre redes inalámbricas, También se describirá en detalle el funcionamiento de NC, haciendo especial hincapié en en la capacidad de recodificar en los nodos intermedios. Posteriormente se hace un pequeño resumen y explicación del funcionamiento de las herramientas de software libre utilizadas para la ejecución de este trabajo, como son la librería *KODO* o la herramienta *Iptables*.

- Capítulo 3: Se explica todo el desarrollo que se ha seguido para conseguir la realización del proyecto. Se enumeran los materiales necesarios para la elaboración del banco de pruebas, así como pequeñas explicaciones de la configuración usada en los dispositivos y del código desarrollado para poder llevar a cabo las simulaciones.

- Capítulo 4: Este apartado se centra en el funcionamiento del simulador que hemos usado con objeto de poder comparar los resultados obtenidos sobre el banco de pruebas con los de la simulación.

- Capítulo 5: Se presentan los resultados obtenidos de las diferentes pruebas contrastándolas con el modelo y los resultados obtenidos en el entorno de simulación. Se evalúan los siguientes parámetros: el tiempo de transmisión, el throughput, etc. Realizamos una comparativa entre los mecanismos clásicos de capa de transporte (TCP y UDP) frente a la solución propuesta, que es una combinación de UDP más NC.

- Capítulo 6: Finalmente se concluye el trabajo resumiendo los principales resultados obtenidos, después se presentan las conclusiones obtenidas del trabajo, además se plantean futuras líneas de uso/investigación para el esquema de red propuesto.

2

Estado del Arte

Este capítulo recoge una introducción a cada uno de los elementos en los que se sustenta el trabajo. De esta forma el lector puede adquirir unos conocimientos básicos que le permitan comprender en mayor medida los diferentes conceptos tratados a lo largo de las siguientes páginas.

2.1. Redes Inalámbricas

Las redes inalámbricas están compuestas por un grupo de nodos los cuales se interconectan sin necesidad de ninguna red cableada, a través de comunicaciones radio. La principal ventaja que ofrecen estas redes es la posibilidad de movilidad sin perder la conectividad, Sin embargo, las redes inalámbricas son propensas a errores debido a las interferencias y movilidad de los nodos. Los dispositivos conectados a una red inalámbrica transmiten a través del medio radio compartido, lo que reduce en gran medida el rendimiento respecto redes cableadas.

Dentro de este tipo de redes podemos distinguir dos grupos claramente diferenciados:

- **Modo infraestructura:** La mayoría de redes a día de hoy son de este tipo, en las que existe un nodo Punto de Acceso Access Point (AP) al que se conectan el resto de dispositivos. Normalmente este dispositivo suele estar conectado al exterior haciendo las funciones de Gateway. Los nodos se comunican exclusivamente con el AP, que más

tarde será el encargado de retransmitir la información. Esto no resulta muy eficiente cuando se desean cubrir áreas muy extensas ya que implica el despliegue de varios APs.

- **Modo ad-hoc:** En este tipo de redes los dispositivos inalámbricos están interconectados entre sí, no existe un nodo central que controle el tráfico. Cada nodo forma parte activa del diseño y mantenimiento de la red. Por otra parte, este tipo de de redes presenta algunos problemas, sobre todo en lo referente al encaminamiento debido a su alta variabilidad en el número de dispositivos, así como en la disponibilidad (cobertura) de los mismos.

Con la topología ad-hoc podemos configurar los nodos para que sean capaces de retransmitir los paquetes de los nodos adyacentes, esto va a derivar en una gran zona de cobertura sin la necesidad de grandes potencias de transmisión por parte de los nodos. Este subtipo de red ad-hoc se denomina (Wireless Mesh Network (WMN)), muy interesantes para su uso en redes de sensores debido a la robustez y a la multitud de rutas posibles que ofrece, así como, la reducción del coste frente a una red en modo infraestructura. Podemos ver un ejemplo de este tipo de redes en la Figura 2.1 [1]

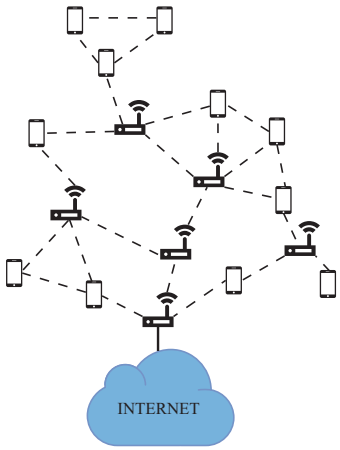

Figura 2.1: Red Inalámbrica con acceso a Internet

Estas redes, a pesar de ser escalables, presentan problemas debido a las interferencias a medida que el número de saltos se incrementa. Por otra parte, está comprobado que el rendimiento de TCP sobre redes inalámbricas no es comparable con su rendimiento sobre redes cableadas [1], ya que el protocolo fue diseñado para redes con muy baja probabilidad de pérdidas, donde la mayor parte son debidas a la congestión ,esta pérdida de rendimiento aumenta a medida que disminuye la calidad del enlace.

[1]Los gráficos con los que han sido elaborados estas imágenes y otras de este trabajo están obtenidas de la página de descarga libre https://www.flaticon.com

Han aparecido numerosas variantes cuyo objetivo ha sido tratar de mejorar el rendimiento en este tipo de redes, cabe destacar el NC entre estos.

En este trabajo se va analizar las técnicas de NC sobre una red inalámbrica en modo Ad-hoc, con objeto de representar fielmente un posible entorno real donde se podría utilizar esta técnica.

2.2. Network Coding

El método "tradicional" de transmisión de información consiste en dividir la información en datagramas y enviarlos por la red desde el nodo origen hasta el nodo destino, donde los nodos intermedios actúan como meros repetidores de la información, aunque puede haber retransmisiones por parte de los nodos intermedios. Estos nodos no manipulan el contenido de los paquetes simplemente retransmiten a nivel de Red.

Hay diversos esquemas que proponen una solución basada en esquemas de codificación como puede ser *LT* o los *Códigos Raptor* [2], sin embargo, estos esquemas solo trabajan con codificación por parte del nodo origen, NC puede aprovechar las capacidades de los nodos intermedios para ofrecer un mayor rendimiento.

Network Coding fue propuesto en el año 2000 en [3], este esquema fue revolucionario en su momento ya que presentaba un planteamiento totalmente diferente de envío de la información, desde ese momento son muchos los esfuerzos que se han dedicado al estudio de las posibilidades de esta técnica.

El objetivo de este estudio, al igual que otros muchos que hay dentro de esta área, es recapitular toda la información posible sobre esta técnica y determinar si realmente es interesante su uso en entornos inalámbricos. También es interesante proponer un primer modelo de protocolo.

Network Coding, como se puede ver en la Figura 2.2 se implementa por encima de la capa de transporte, de esta forma conseguimos un envió seguro de la información extremo a extremo utilizando UDP. Esto es así salvo en el caso que usemos *recoding*, técnica que se explica en el apartado 2.4, en ese esquema los nodos intermedios realizan las operaciones por debajo de la capa de transporte, mientras que los nodos origen y destino siguen apoyándose en UDP para el envío de la información.

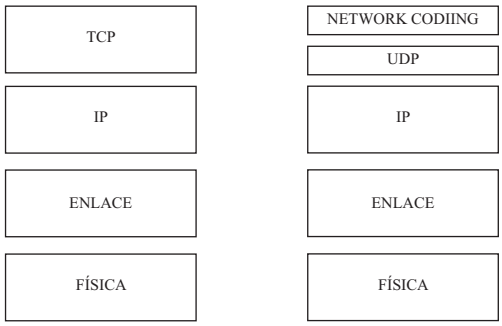

Figura 2.2: Estructura en capas con NC

Dentro de estas técnicas podemos diferenciar claramente 2 en función de cual sean los flujos de información que deseamos codificar, podemos codificar la información propia

de nuestro dispositivo, *Intra-flujo*, o por el contrario podemos tratar de optimizar el envío de información sobre medios broadcast, *Inter-flujo*.

2.2.1. Inter-flujo

En este esquema, como se puede ver en la Figura 2.3 existe un nodo intermedio *Coding Node* que hace las labores de "concentrador" del tráfico, es el encargado de combinar los diferentes paquetes provenientes de flujos independientes. Este "paquete" combinado se retransmite a todos los nodos receptores, esta técnica aprovecha el medio radio, el cual es compartido.

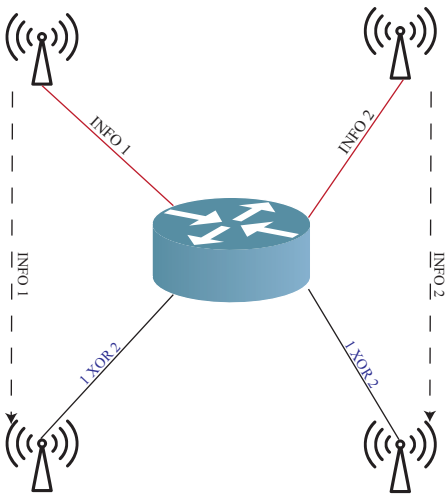

Figura 2.3: Inter-Flujo NC

Para que este proceso se lleve a cabo hay que tener algunos aspectos en cuenta:

- **Tiempo:** No todos los paquetes van a llegar de manera simultánea al *Coding Node* por lo que tendremos que establecer un temporizador, C_t que mantiene un buffer en el Coding Node donde se almacenaran los paquetes recibidos en ese periodo.
- **Tamaño:** Debemos establecer un tamaño máximo del buffer B_s con objeto de no enviar paquetes de un tamaño muy elevado. Si el temporizador o el tamaño máximo se sobrepasa el primer paquete se enviará de manera "tradicional".
- **Decodificadores:** Los nodos finales tienen implementado *Overhearing* es decir tienen que escuchar los flujos de información que no van dirigidos a ellos, los cuales son necesarios para poder decodificar el paquete correctamente, para esta labor también necesitan tener implementado un buffer.

Mediante NC se puede reducir el número de paquetes enviados, cuanto mayor sea el grupo de paquetes codificados. El procedimiento para conseguir esto es realizar una operación *XOR* a nivel de bit entre todos los paquetes que queramos agrupar, esta operación se realiza en el *Coding Node*.

Como podemos ver en, este último es el que realiza las operaciones y retransmite en *"broadcast"* el paquete resultante a los nodos destino ($P = P_1 \oplus P_2 \oplus P_3$). De esta forma conseguimos enviar toda la información con 4 transmisiones, en lugar de las 6 que serían necesarias mediante el método çlásico".

El proceso de decodificación se basa en realizar de nuevo la operación XOR con el paquete recibido y con los paquetes de los demás nodos, los cuales los ha recibido mediante el *overhearing* aprovechando las características del medio radio que es compartido, de esta forma obtenemos la información destinada a este nodo. Por ejemplo, para el caso del flujo de información *1* $P_1 = P_c \oplus P_2 \oplus P_3 = (P_1 \oplus P_2 \oplus P_3) \oplus P_2 \oplus P_3$.

Así conseguimos reducir el número de transmisiones lo cual presenta diferentes ventajas como: mejora de la eficiencia energética, reducción de la congestión de la red, etc.

2.2.2. Intra-flujo

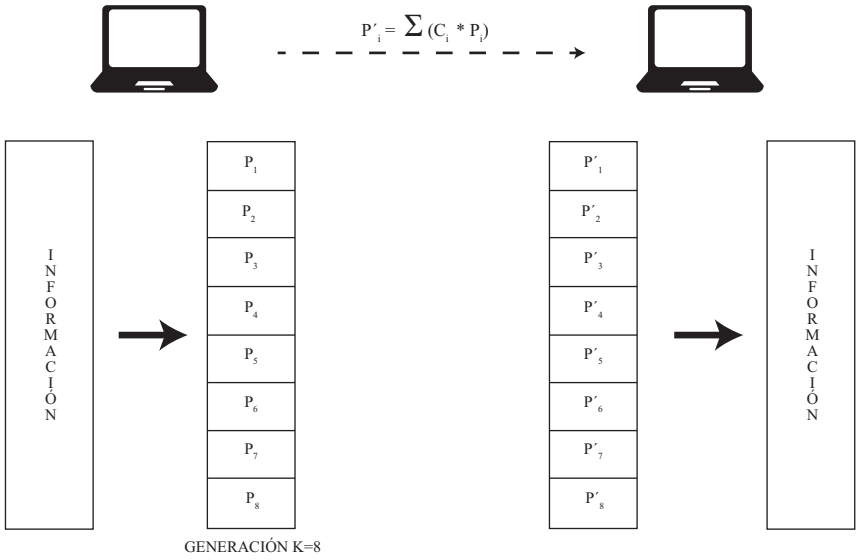

Figura 2.4: Intra-Flujo NC

En este apartado vamos a explicar la técnica que se usa en este trabajo el *Network Coding Intra-Flujo*, consiste en "combinar" los flujos de información pero en este caso provenientes de un mismo nodo, esta técnica fue descrita por primera vez por *Chachulski, Jennings, ...* en [4], consiste en una técnica llamada *MAC-independent opportunistic routing protocol (MORE)*, donde se plantea el uso de este tipo de técnicas sobre enlaces inalámbricos con grandes pérdidas implementado sobre *Roofnet*, una implementación (WMN) en la cual se trata de suministrar cobertura inalámbrica a través de dispositivos distribuidos por el área sin conexión cableada.

Lo primero que debemos hacer es especificar los parámetros con los que vamos a trabajar para tener una idea clara del proceso.

Termino	Descripción
Paquete Nativo	Paquete sin aplicar Network Coding
Paquete Codificado	Combinación lineal de los fragmentos del paquete original
Vector de coeficientes	Vector donde se indica cómo obtener el paquete original a partir de los decodificados. $\overrightarrow{C_j} = (C_{j1}, C_{j2}, ..., C_{jk})$
Paquete innovador	Un paquete es innovador si es linealmente independiente de los recibidos anteriormente.
GF(Q)	Coeficientes del cuerpo de *Galois*

Tabla 2.1: Parámetros básicos de transmisiones Intra-flow [4]

Con las técnicas Intra-flujo, la información se divide en fragmentos, el número total de fragmentos lo vamos a denominar K. Después se realizar una combinación lineal aleatoria de los fragmentos, Figura 2.4, el transmisor genera los paquetes codificados, que serán los que posteriormente serán enviados por el medio radio. El nodo transmisor es el encargado de realizar las operaciones de codificación sobre la información.

Posteriormente el nodo receptor debe ir almacenando los paquetes codificados hasta ser capaz de conseguir decodificar correctamente la información, para lograr este proceso necesitara al menos K paquetes codificados. El nodo destino es capaz de decodificar la información gracias a la información que contiene el vector de codificación, donde se encuentran los coeficientes usados para generar cada uno de los paquetes codificados.

Para la decodificación los nodos deben obtener K paquetes linealmente independientes para esto los nodos hacen uso de la eliminación Gaussiana, usando la información del vector de coeficientes.

Para poder realizar tanto la codificación como la decodificación es necesario sendos búferes en el nodo origen y en el nodo destino.

Existen diferentes tipos de codificación de los paquetes nosotros vamos a usar RLNC, aunque existen otras variantes como Tunable Sparse Network Coding (TSNC) orientado a reducir el coste computacional, en esta variante los paquetes que se usan para codificar son un sub-conjunto del total de paquetes (K), de esta forma se reduce el tiempo de decodificación aunque la probabilidad de enviar un paquete linealmente independiente disminuye.

Estas técnicas puedes ser implementadas tanto para comunicaciones unicast (En las que nos vamos a centrar) como para multicast, donde diversos trabajos han demostrado que también son de gran utilidad.

El desestimar el uso de TCP nos soluciona uno de los mayores problemas que acarreaba este protocolo el mecanismo *Congestion Avoidance* el cual considera la pérdida de paquetes como una congestión de la red, reduciendo el número de paquetes transmitidos para solucionar esta situación que el protocolo interpreta como congestión. Por otra parte los paquetes en TCP están ordenados mediante su número de secuencia, la pérdida de un paquete puede provocar la retransmisión de varios paquetes, con la bajada de rendimiento que eso implica.

Vamos a analizar la idea de que los nodos intermedios pueden transmitir paquetes recodificados, en vez de retransmitir los paquetes codificados recibidos.

2.3. Random Linear Network Coding (RLNC)

Dentro de las técnicas de NC podemos encontrar RLNC, el cual fue propuesto en [5], en este tipo de NC los paquetes codificados se generan a través de la combinación lineal aleatoria de la información.

Este trabajo está basado en otros trabajos anteriores donde ya se comenzó a trabajar con este tipo de despliegues ([6][7], [8], [9]). Como podemos ver en la Figura 2.4 la información es dividida en, *generaciones*, a su vez cada generación es dividida en K paquetes. En el nodo origen existe un buffer con estos paquetes los cuales son multiplicados por un factor aleatorio y sumados entre sí, el resultado de esta operación será el enviado. $\sum_{i=0}^{K-1} = C_i * P_i$. Los coeficientes C_i son elementos del cuerpo de Galois, $GF(Q) = GF(2^q)$, por otra parte, los P_i son los fragmentos en los que se ha dividido la información. Podemos implementar estos de dos formas, el nodo transmisor envía paquetes hasta que reciba una confirmación o podemos simplemente enviar mensajes sin esperar confirmación, sería conveniente enviar un numero de paquetes con un margen adecuado para que se pueda decodificar la información de forma correcta. Para la correcta decodificación son necesarias dos matrices, la primera matriz ($\mathbf{C}$) $K \times K$ almacena los vectores de coeficientes ($\overrightarrow{C_j} = (C_{j1}, C_{j2}, ..., C_{jk})$) de cada uno de los paquetes recibidos. La segunda matriz almacena los paquetes codificados recibidos útiles para la decodificación.

Cada paquete recibido *"contiene"* ($\frac{1}{K}$) información, con una peculiaridad no importa si se pierde un paquete concreto, ya que esta pérdida se suplirá con otro paquete. Para lograr poder decodificar la generación son necesarios K paquetes innovadores. Cada vez que se recibe un paquete se insertan los coeficientes en las matriz de decodificación y se comprueba si es linealmente dependiente, esto lo podemos comprobar fácilmente calculando el rango de la matriz. En caso de que el rango aumente significa que el paquete es innovador y se procede a guardar el contenido del mismo. Una vez la matriz ha llegado a rango K significa que ya tenemos toda la información necesaria, además podemos realizar la inversa de la matriz (C^{-1}) con la que conseguiremos poder obtener los paquete originales. ($P_{orig} = C^{-1} \cdot P'$).

En el caso de que estemos en un modelo de transmisión con reconocimientos el receptor enviará el Acknowledgement (ACK) cuando el rango sea igual a K, el emisor no parará de enviar paquetes hasta que reciba el ACK. El número de paquetes codificados que son necesarios enviar siempre va a ser mayor o igual a K ya que no todos los paquetes llevaran información útil.

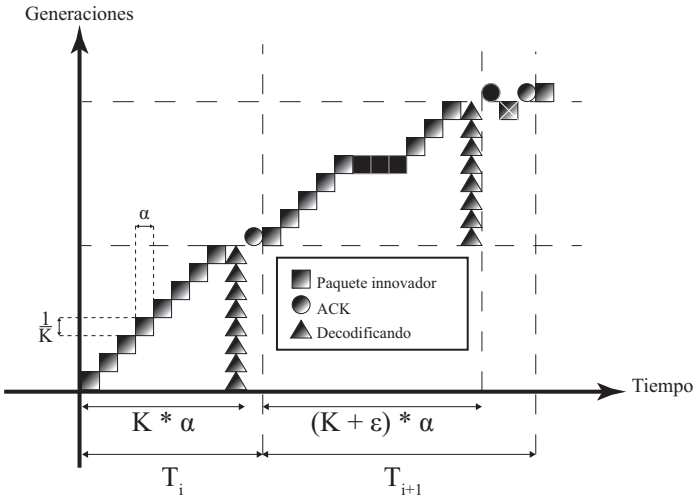

Figura 2.5: Transmisión RLNC (K=8)

En la Figura 2.5 podemos ver la transmisión de dos generaciones basadas en RLNC con un número de símbolos igual a 8 (K = 8), la primera generación podemos apreciar que es una transmisión ideal, es decir todos los paquetes son innovadores(son linealmente independientes entre sí), además no se producen pérdidas de paquetes. El tiempo que tarda en recibir el destino la generación T_i es igual al tiempo por símbolo τ multiplicado por el número de símbolos en la generación. También estamos representando el tiempo de computo necesario para decodificar el mensaje (calcular C^{-1} y multiplicarla por los paquetes recibidos), así como el envío de un ACK de confirmación al emisor. En la transmisión de la segunda generación podemos ver cómo además de los paquetes innovadores recibimos paquetes linealmente dependientes a los coeficientes que ya se encuentran en C, a este conjunto de paquetes los vamos a llamar ε, en este caso el tiempo de transmisión se puede calcular como $T_{trans} = (K + \varepsilon) \cdot \tau$. Podemos observar en la segunda generación como se pierde el paquete de confirmación, recordemos que el nodo origen mandara paquetes hasta que reciba esa confirmación con lo cual, la pérdida de ese ACK está introduciendo en la red paquetes innecesarios, el ACK se vuelve a enviar y se da por concluido el envío.

Para el correcto funcionamiento tenemos implementada la cabecera propietaria que podéis ver en la Figura 2.6, en ella podemos ver dos partes claramente diferenciadas. La primera de ellas es de longitud fija (9 Bytes) y contiene información necesaria para la correcta transmisión entre los nodos. Analicemos un poco más en detalle cada uno de los campos que componen la cabecera, Figura 2.6:

- *Type:* Identifica si el paquete es de datos o un reconocimiento.
- *K:* Representa el número de paquetes en cada generación. Diferentes parámetros como el Throughput, ya que el número de paquetes linealmente independientes que recibe el receptor depende de K, El tamaño del buffer en los nodos debe ser de al

menos K paquetes, por lo que aumentar el tamaño de la generación también implica un mayor tamaño de los buffers y del tiempo de computación.

- *q:* Este valor indica el número de bits con los que se compone cada uno de los coeficientes C_i. Estos coeficientes se generan a partir de un cuerpo finito de la forma: $GF(Q) = GF(2^q)$. Este parámetro influye también en el rendimiento computacional de forma más compleja, en la implementación usada trabajaremos con (q=1), es decir los valores de los coeficientes son binarios (pueden ser '0' ó '1').

- *Frag:* Este valor nos indica de que generación es ese paquete codificado, permitiéndonos incluso poder trabajar con varias generaciones simultáneamente[2], lo cual puede llegar a ocurrir, aunque de forma bastante improbable.

- *Puertos:* Es necesario el uso de puertos(al igual que en las comunicaciones clásicas) para poder distinguir correctamente los flujos de información generados por las distintas aplicaciones en un dispositivo

- *Vector $\overrightarrow{C_j}$:* Es el vector de codificación con el que se ha generado el correspondiente paquete. Como se ha mencionado, es necesario incluirlo en la cabecera para posibilitar la decodificación, en el nodo receptor se introduce en la matriz C, y se comprueba si es linealmente independiente, con objeto de determinar si es un paquete innovador o no.

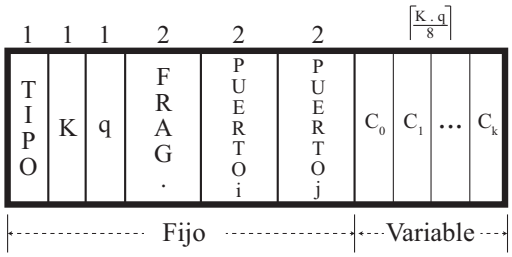

Figura 2.6: Cabecera NC

Aunque vamos a presentar un modelo teórico con un solo nodo intermedio se pueden extrapolar los resultados a transmisiones con un número mayor de saltos.

[2]Si recibimos un paquete de otra generación completada lo descartamos

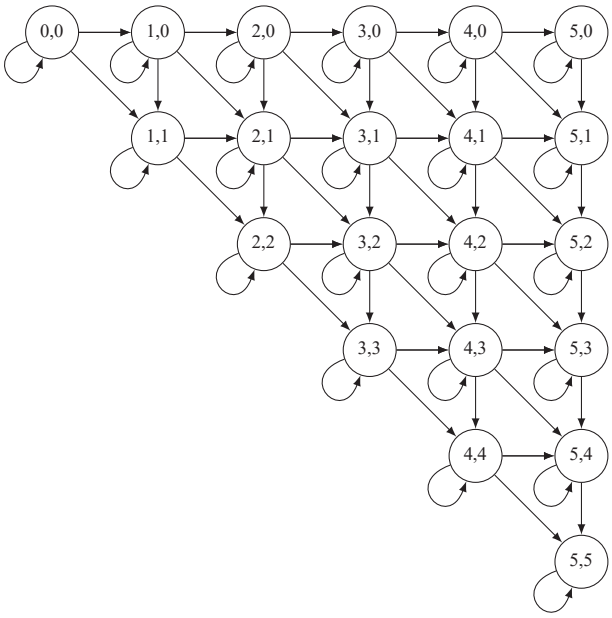

Figura 2.7: Cadena de Márkov absorbente

El modelo analítico que se va a exponer brevemente se basa en el trabajo [2]. Cada estado de la cadena está representado por la dupla *(r,d)* donde r corresponde con el rango en el nodo intermedio y d el rango del nodo destino, como es lógico el rango del receptor (R) nunca va a ser mayor que el rango del nodo intermedio (D). El nodo *(k,k)* se corresponde con el único estado absorbente de la cadena, puesto que el rango, tanto en el nodo intermedio como en el nodo final, nunca va a decrementar; además una vez que el rango del receptor llegue a k será capaz de decodificar correctamente la generación.

En el ejemplo que vemos de manera gráfica en la Figura 2.7 tenemos una generación compuesta por cinco símbolos (K = 5). La probabilidad de que llegue un paquete linealmente dependiente se puede calcular a partir del trabajo de Trullols [10] se puede interpretar como la probabilidad de escoger una combinación de los paquetes de entre las que ya posee el nodo Q^r, del total de posibles combinaciones Q^k. Por lo tanto, la probabilidad de que aumente el rango en el nodo intermedio es: $\phi_r = 1 - \frac{Q^r}{Q^k}$, donde Q es el campo de Galois utilizado ($GF(Q = 2^q)$).

En el caso del nodo destino la expresión es muy similar, la única diferencia es que el total de combinaciones linealmente independientes está determinado por el rango del nodo intermedio. $\phi_d = 1 - \frac{Q^r}{Q^d}$. Tenemos que tener en cuenta que el nodo intermedio solo reenvía los paquetes del nodo origen, por lo que si no recibe paquetes tampoco enviará nada al nodo destino.

Está claro que el rango de cualquiera de los nodos solo puede incrementarse en uno como mucho por cada transmisión recibida. Por esto, la probabilidad de transición $\pi(i, j)$

$$\pi(i,j) = \begin{cases} p_1 + (1-p_1)\left(\frac{Q^r}{Q^k}\right)p_2 + (1-p_1)\left(\frac{Q^r}{Q^k}\right)(1-p_2)\left(\frac{Q^d}{Q^r}\right) & i=0, j=0 \\ (1-p_1)\left(1-\frac{Q^r}{Q^k}\right)p_2 & i=1, j=0 \\ (1-p_1)\left(\frac{Q^r}{Q^k}\right)(1-p_2)\left(1-\frac{Q^d}{Q^r}\right) & i=0, j=1 \\ (1-p_1)\left(1-\frac{Q^r}{Q^k}\right)(1-p_2)\left(1-\frac{Q^d}{Q^{r+1}}\right) & i=1, j=1 \end{cases} \tag{2.1}$$

Figura 2.8: Probabilidades sin Recoding

de un estado concreto a otro es $(r,d) \rightarrow (r+i, d+j)$, donde $(i,j)\epsilon[0,1]$, lo que se resume en 4 posibles casos.

Podemos observar las diferentes probabilidades de transito de la cadena de Márkov en la Figura 2.8.

2.4. Recoding

Hasta ahora hemos estado hablando sobre como enviar paquetes codificados de un nodo origen a un nodo destino, cuando el nodo intermedio meramente retransmitía la información. Es cierto que se ha mencionado que los nodos intermedios pueden hacer *Forwarding* de los paquetes recibidos, pero en ningún momento forman parte "activa" en el proceso de codificación.

Vamos a analizar otra situación, el nodo origen y destino hacen lo mismo que en el caso anterior pero los nodos intermedios captan la información codificada recibida por el nodo origen y aplican técnicas de codificación adicionales (*Recoding*).

Hasta la fecha no hay una respuesta precisa sobre cuando es recomendable que los nodos intermedios hagan *recoding* o simplemente reenvíen los paquetes que reciben. A través de este informe y sus resultados se aporta más información acerca este parámetro, además de otros resultados.

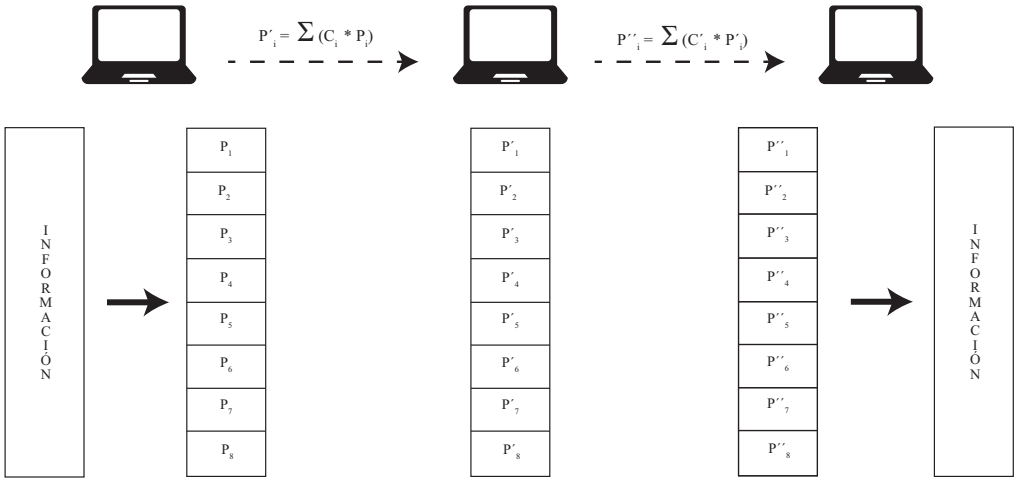

Figura 2.9: Esquema Recoding

Como podemos ver en la Figura 2.9 en este caso el nodo intermedio almacena los paquetes codificados innovadores debido a esto es necesario también que guarde la información pertinente (buffer, matriz C), al igual que hace el receptor en el primer ejemplo.

Sin embargo, ahora el nodo intermedio no retransmite los paquetes recibidos sino que realiza un nuevo proceso de codificación sobre los paquetes que hemos recibido, expresado matemáticamente tendría la siguiente forma $P'' = \sum_{i=0}^{K-1} = C'_i * P'_i$. Estos paquetes son diferentes de los paquetes codificados por el emisor, con lo cual su vector de coeficientes también es diferente, $\overrightarrow{C'_j} = (C'_{j1}, C'_{j2}, ..., C'_{jk})$.

Vamos a volver a basarnos en el modelo de la Figura 2.7, los estados de la cadena de

$$\pi(i,j) = \begin{cases} \alpha_1\alpha_2 + \alpha_1(1-\alpha_2)\left(\frac{Q^d}{Q^r}\right) + (1-\alpha_1)\left(\frac{Q^r}{Q^k}\right)\alpha_2 + \\ \quad + (1-\alpha_1)\left(\frac{Q^r}{Q^k}\right)(1-\alpha_2)\left(\frac{Q^d}{Q^r}\right) & i=0, j=0 \\ (1-\alpha_1)\left(1-\frac{Q^r}{Q^k}\right)\alpha_2 + \\ \quad + (1-\alpha_1)\left(1-\frac{Q^r}{Q^k}\right)(1-\alpha_2)\left(\frac{Q^d}{Q^{r+1}}\right) & i=1, j=0 \\ \alpha_1(1-\alpha_2)\left(1-\frac{Q^d}{Q^r}\right) + \\ \quad + (1-\alpha_1)\left(\frac{Q^r}{Q^k}\right)(1-\alpha_2)\left(1-\frac{Q^d}{Q^r}\right) & i=0, j=1 \\ (1-\alpha_1)\left(1-\frac{Q^r}{Q^k}\right)(1-\alpha_2)\left(1-\frac{Q^d}{Q^{r+1}}\right) & i=1, j=1 \end{cases} \tag{2.2}$$

Figura 2.10: Probabilidades usando Recoding

Márkov se mantienen iguales, mientras que las probabilidades de transición entre estados cambia.

Como ya hemos dicho antes, la probabilidad de que aumente el rango se basa en la llegada de un paquete innovador para el nodo intermedio. La probabilidad será la misma que en el modelo anterior, $\phi_r = 1 - \frac{Q^r}{Q^k}$. En el caso del nodo destino la diferencia es que el total de combinaciones linealmente independientes está determinado por el rango del nodo intermedio, $\phi_d = 1 - \frac{Q^r}{Q^d}$, además en esta configuración el nodo intermedio va a generar paquetes a pesar de no recibir ningún paquete del nodo origen.

Las probabilidades de transición entre los estados van a variar respecto al modelo anterior, las diferentes probabilidades de transición entre los estados de la cadena se pueden ver en la Figura 2.10

Bajo ciertas condiciones, es probable que el nodo intermedio genere un mayor número de dependencias lineales, es decir, sin información útil para el destino. Sobre todo el inicio de la transmisión, cuando el nodo intermedio ha recibido solo unos pocos paquetes codificados. Por eso es interesante establecer un umbral a partir del cual el nodo intermedio empiece a recodificar, cuando el rango de la matriz C del nodo intermedio supere el valor del umbral enviará paquetes recodificados ($C >= Thres$). En el apartado de resultados vamos a tratar de establecer cuál es el umbral óptimo para las transmisiones, los valores de *Threshold* se obtienen evaluando el modelo teórico para todos los valores posibles.

Ahora que ya hemos caracterizado las probabilidades de llegada de un paquete linealmente independiente, tanto sin uso de Recoding como con su uso, vamos a aplicar las propiedades características de las cadenas de Márkov absorbentes [11] que nos permitirán obtener más información útil acerca del sistema.

Una vez que tenemos las probabilidades de transición entre los estados podemos hallar la matriz fundamental, aquí podemos ver su forma canónica 2.3 con t estados transitorios y a estados absorbentes. I se trata de la matriz identidad, ya que la probabilidad de permanecer en el estado absorbente es '1', $Q_{t\times t}$ expresa la probabilidad de ir del estado i al estado j, por último $R_{t\times a}$ expresa la probabilidad de ir del estado transitorio t al estado absorbente a en este caso es un vector columna ya que solo hay un estado absorbente.

$$\mathcal{P} = \begin{bmatrix} I_{a\times a} & 0 \\ R_{t\times a} & Q_{t\times t} \end{bmatrix} \tag{2.3}$$

Vamos a hacer uso de uno de los teoremas propuestos en [11]

Teorema 1. *Número medio de transmisiones: El número medio de transmisiones*[3] *antes de llegar al estado absorbente empezando en el estado x es el elemento x-ésimo del vector M 2.4*

$$M = (I - Q)^{-1}\,\Delta \tag{2.4}$$

I es una matriz identidad de la misma dimensión que Q, y Δ es un vector columna con todos sus componentes a '1'. El primer elemento de M indica el número medio de transmisiones necesarias para poder completar la transmisión.

[3]Solo se están teniendo en cuenta las transmisiones que tiene que realizar el nodo emisor, en este ejemplo una transición de estado involucra dos transmisiones: Emisor → Nodo intermedio, Nodo intermedio → Receptor

2.5. UDP

Como ya hemos indicado anteriormente se ha propuesto el uso de NC junto con UDP. Es un protocolo de capa de transporte (*Capa 4*) el cual se basa en el intercambio de datagramas sin establecimiento de la conexión entre el nodo origen y el nodo destino. Cabe destacar de este protocolo que realiza entregas no fiables, es decir no está garantizada la recepción correcta del paquete como si ocurre en el caso de TCP. Esto es debido a que no se realiza control de errores ni implementa retransmisiones además tampoco posee control de flujos y los paquetes pueden llegar desordenados. NC se encargará de la entrega fiable y ordenada de los paquetes.

Por otra parte, la velocidad de transmisión que ofrece UDP es superior a la de TCP, esto puede ser interesante cuando la la pérdida de un datagrama no es importante para que la aplicación siga funcionando. Algunos protocolos que trabajan sobre UDP son Dynamic Host Configuration Protocol (DHCP) o Domain Name System (DNS). También es interesante su uso sobre plataformas multimedia en tiempo real, tanto para vídeo como para audio, en estos casos los requisitos de retardo entre paquetes son muy exigente, además el hecho de perder un datagrama de información no es un fallo critico que impida la comunicación.

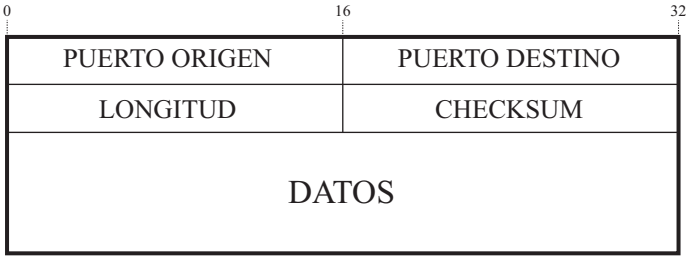

Figura 2.11: Cabecera UDP

Aquí podemos ver el formato de la cabecera de UDP, (Figura 2.11), es un formato de cabecera bastante simple debido a que no implementa apenas funciones. La función de los diferentes campos de la cabecera es la siguiente:

- Puerto origen: El campo está compuesto por 16 bits, al igual que todos los restantes de la cabecera. Identifica el puerto origen desde el que se realiza la comunicación, este campo se utiliza para multiplexar comunicaciones de diferentes aplicaciones provenientes de un mismo equipo.
- Puerto destino: Sirve para identificar la aplicación que va a recibir el mensaje en el dispositivo.
- Longitud: Indica la suma total de los bytes incluyendo la cabecera UDP más el campo de datos.

- Checksum: Hace una comprobación sobre la cabecera UDP, una pseudo-cabecera IP y el campo de datos. Si el número de bits no es múltiplo de 16 de rellena con 0s.

En nuestro caso todas las carencias que presenta UDP las vamos a subsanar con el uso de Network Coding. Ofreciendo un sistema de entrega ordenada y fiable.

2.6. Librería KODO

KODO [12] es una librería de código abierto escrita en C++. Está orientada con fines educativos y de investigación. Se compone de diferentes algoritmos los cuales pueden ser utilizados para implementar técnicas de Network Coding, estos algoritmos se encuentran implementados en pequeños bloques con objeto de poder ser usados de manera sencilla. La Application Programming Interface (API) que aporta KODO facilita en gran medida la creación de programas ya que permite a los programadores abstraerse de la implementación del algoritmo como tal y poder centrarse en otros aspectos. La idea de este proyecto es que los investigadores usen y modifiquen el código de esta librería para evaluar las técnicas de NC o proponer sistemas como es el caso de este trabajo.

Como se indica en ([13] , [9]) esta librería presenta numerosas opciones y formas de codificación. Primero de todo podemos elegir entre diferentes códecs para NC como puede ser: RLNC, Systematic RLNC o Sparse RLNC. Destacan la codificación *systematic* que consiste en enviar en primer lugar los paquetes sin codificar, después envía los paquetes codificados con objeto de recuperar los paquetes perdidos durante la transmisión. Otro tipo importante de codificación es *On-the-fly coding* en este caso el emisor transmite los datos según se va generando la información, no es necesario tener toda la información para poder enviar paquetes como en el esquema tradicional que es el que usamos nosotros.

También mencionar que el uso de NC se puede acompañar de otros códecs como puede ser los códigos Reed-Solomon(RFC 5510)[14], Carousel code(RFC 5445) o Random Annex overlay code entre otros.

Un aspecto muy importante de esta librería es la cantidad de dispositivos que son compatibles con ella, lo cual es muy útil como en nuestro caso que usamos Raspberry PIs. Además, se puede utilizar las herramientas de *cross-compilation*, que permite generar los binarios ejecutables en un servidor. Esto permite compilar el proyecto, un proceso bastante costoso, en entornos con mayor capacidad de computación. En nuestro caso el compilar la librería en la Raspberry implicaría mucho tiempo, por eso realizamos la compilación desde nuestro propio PC.

Por último, mencionar que existe una API en Python para poder utilizar dicha librería. Sin embargo, no es tan potente y no permite alterar los algoritmos de codificación/decodificación.

2.7. Iptables/Netfilter

Netfilter[15] es un *Framework* disponible para Linux el cual presenta numerosas utilidades como crear cortafuegos a través del filtrado de paquetes. También permite utilizar, traducción de direcciones de red (NAT), crear archivos *log* con estadísticas de los paquetes recibidos... Para la captura de paquetes, que es el caso que a nosotros nos interesa en este caso, la herramienta permite diferentes manipulaciones de los paquetes se pueden interceptar, manipular, eliminar...

Lo primero que debemos saber es que cada protocolo tiene establecidos diferentes *"hooks"* que son puntos bien-conocidos, por ejemplo, para el caso de IP existen 5 puntos donde se pueden capturar los paquetes, más tarde hablare de ellos. Básicamente lo que hace Netfilter es cuando un paquete pasa por alguno de los *"hooks"*, comprueba que debe realizar con él, esto se establece mediante reglas que explicaré en la parte de desarrollo. La herramienta más conocida de este paquete es **Iptables** la cual se pensó para hacer funciones de Firewall pero en nuestro caso nos va a resultar muy útil para poder manipular los paquetes en el nodo intermedio sin necesidad de tener que pasar por todas las capas del nivel Open System Interconnection (OSI), podemos capturar los paquetes íntegros a nivel de red.

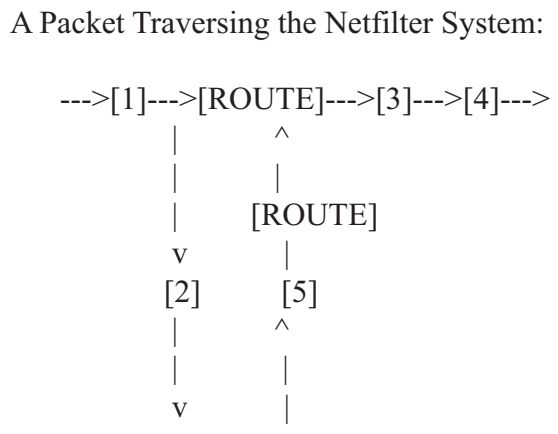

Figura 2.12: Netfilter hooks

Como podemos ver en la Figura 2.12 en el caso de IP existen 5 puntos donde se puede capturar el paquete que vamos a explicar en más detalle:

- NF_IP_PRE_ROUTING (Hook 1): En si son los paquetes apenas han llegado al dispositivo sin manipular, las únicas comprobaciones que se realizan es el IP Checksum, y algunos otros aspectos básicos.

- NF_IP_LOCAL_IN (Hook 2): Tras este proceso pasa por el código de enrutamiento que decide si el paquete va a destinado a otra interfaz o hacia un proceso local, en el caso de que no esté establecido el forwarding en el nodo los paquetes dirigidos hacia otro destino se descartaran. En caso de que el paquete sea dirigido hacia el propio nodo entonces es cuando pasa por este gancho.
- NF_IP_FORWARD (Hook 3): Tras el enrutamiento anterior si la solución se basa en dirigirlo a otra interfaz entonces es este punto el que podrá capturar el paquete.
- NF_IP_POST_ROUTING (Hook 4): Gancho que se encuentra justo antes de la salida del paquete del dispositivo es el punto más cercano a la salida del paquete, en muchos casos el paquete que se puede capturar en **3** y en **4** es el mismo.
- NF_IP_LOCAL_OUT (Hook 5): Por este punto pasan los paquetes generados localmente antes de ser dirigidos por el proceso de routing.

Una vez que el paquete pasa por el nodo sobre el que deseamos actuar Netfilter presenta diversas acciones para realizar sobre el paquete:

- NF_ACCEPT: Deja pasar el paquete de modo normal sin realizar ninguna acción sobre él.
- NF_DROP: Todo lo contrario, tira el paquete interrumpiendo la comunicación.
- NF_STOLEN: Captura el paquete para si mismo, evitando en muchos casos que siga su ruta habitual.
- NF_QUEUE: Traslada el paquete a una cola para ser procesado más tarde por el nivel de usuario.
- NF_REPEAT: Llama de nuevo a este gancho.

3

Implementación de la plataforma y del simulador

En este capítulo vamos a explicar detalladamente el proceso llevado a cabo para realizar el despliegue de la plataforma, compuesto por la configuración de los dispositivos y de la elaboración de los programas utilizados, así como explicar brevemente el uso y funcionamiento del simulador *ns-3* con el que vamos a complementar los resultados obtenidos en la plataforma.

3.1. Despliegue de las Raspberries Pi

Se va a detallar el despliegue del benchmark de experimentación utilizado (Figura 3.1). Se explicará toda la configuración llevada a cabo en las Raspberry Pi para su correcto funcionamiento, también se hará una breve descripción del código desarrollado para poder realizar las simulaciones necesarias, así como de la forma de extraer los datos deseados de las mismas.

Figura 3.1: Panel con Raspberries PI

Además, se describirá brevemente el uso del simulador *ns-3* el cual también ha sido utilizado con objeto de contrastar y extender los resultados obtenidos mediante la plataforma real.

3.1.1. Despliegue

Para el despliegue del banco de pruebas utilizado para obtener los resultados (Figura 3.1) de este trabajo han sido necesarios numerosos materiales:

- 20 Raspberry Pi.
- 23 Cable Ethernet para conectar las Raspberries con los switches.
- 20 Tarjetas SD con almacenamiento suficiente para el sistema operativo
- 2 Switches para aportar conectividad a las Raspberries
- 1 Router Wi-fi que permite la comunicación de las Raspberries con el exterior.
- 20 pantallas necesarias para mostrar gráficamente el intercambio de información entre los nodos.
- 20 Cables de alimentación para las Raspberries Pi.
- Panel con superficie suficiente para fijar los dispositivos a ella.
- 5 Regletas para poder alimentar las Raspberries

- Módulo USB Wi-fi para conectar el PC con el router que da conectividad a las Raspberries
- Tornillos para fijar las placas (4 por placa).
- Tuercas para fijar las placas (4 por placa).
- Arandelas de goma aislantes para fijar las placas (4 por placa).

3.1.2. Configuración de los dispositivos

Para poder facilitar la manipulación y el uso de los dispositivos vamos a trabajar en todo momento desde un ordenador desde el cual vamos a poder controlar remotamente los dispositivos. De esta forma nuestro PC se conectará con el router que funciona como *Gateway* para el TestBed a través de esta conexión vamos a poder manipular fácilmente las Raspberry PI además de poder extraer los resultados fácilmente.

Primero se debe definir la configuración de la red[1] de nuestra red. Primero vamos a explicar la red cableada, cada Raspberry es conectada mediante cable Ethernet a su correspondiente switch.

Estos a su vez se conectan a un router Wi-fi, el cual ofrece salida a Internet. Esta salida a la red nos permitirá fácilmente actualizar los dispositivos, descargar código del repositorio Git,etc.

Por otra parte aprovechamos la interfaz Wi-fi que posee el router para establecer conexión directa con nuestro PC, desde el que tomaremos control total sobre las placas.

Indicar que en las Raspberries está instalada Raspbian una distribución Linux especialmente orientada a estos dispositivos, lo que nos va a permitir usar muchas de las herramientas y comandos propios de cualquier SO operativo Linux.

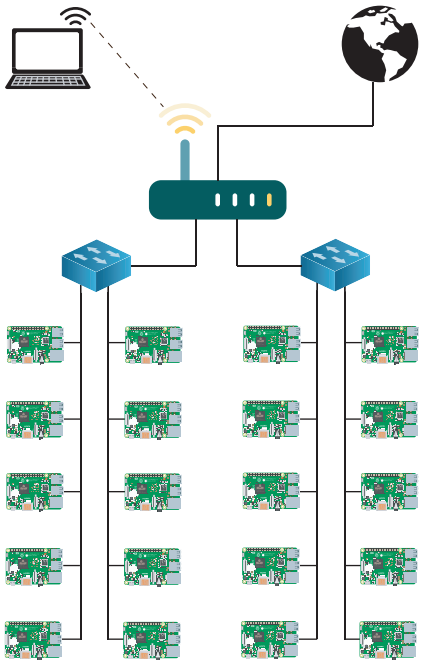

Figura 3.2: Distribución panel

[1]En el esquema la topología esta simplificada, en realidad cada Raspberry tiene una conexión punto a punto con el switch correspondiente.

Para realizar esta configuración, se puede ver en la figura 3.2 vamos a definir las IPs de las Raspberry de forma estática de acuerdo a su posición en el panel, empezando por la esquina superior izquierda y aumento en dirección descendente, esto lo realizamos mediante la configuración DHCP del router, aunque también puede realizarse a través del fichero de configuración de las interfaces propio de Linux (*/etc/network/interfaces*), aunque sería necesario modificar uno a uno siendo mucho más lento.

Por otra parte, las Raspberries Pi forman entre todas ellas una red *ad-hoc*, como ya he explicado anteriormente en la Sección 2.1 las redes ad-hoc están descentralizadas y no tienen un nodo central que conmuta los paquetes, en esta red todos los nodos forman parte activa del envío de la comunicación.

Para la correcta comunicación entre los nodos todos ellos deben estar en la misma celda, al igual que en el caso anterior vamos a definir las direcciones IP de manera estática, a través de (*/etc/network/interfaces*), con objeto de facilitar las configuraciones posteriores sobre la red, como pueden ser las tablas de rutas.

Una vez que ya se encuentra definida correctamente la topología de la red es la hora de poder conectarse a las Raspberries Pi, para ello vamos a hacer uso de *Secure Shell* (SSH) que es un protocolo mediante el cual podemos conectarnos a otro dispositivo y tomar control sobre él. Para poder usar este protocolo necesitamos hacer uso de una terminal Linux, hoy en día se puede integrar directamente en Windows esta consola, o podemos hacer un uso de un software específico, como puede ser Putty o XShell.

También es necesario realizar intercambios de archivos entre el PC y las Raspberry-PIs para ello usamos el protocolo Secure File Transfer Protocol (SFTP), el cual se puede usar mediante el software Xftp, además de otras muchas opciones.

Para poder conectarse a los dispositivos a través de este protocolo lo podemos hacer mediante el uso de un usuario o contraseña o mediante el uso de claves SSH (Criptografía asimétrica), empezaremos utilizando la primera para acceder a las Raspberries pero para la configuración de las mismas de forma automática utilizaremos una pareja de claves pública/privada, de esta forma evitaremos tener que introducir usuario y contraseña en cada uno de los dispositivos que configuremos.

Linux ya incorpora una herramienta para poder crear parejas de claves *ssh-keygen -t rsa* la opción "-t" especifica el algoritmo criptográfico que queremos usar, existen otras muchas opciones para este comando como puede ser la longitud de la clave o usar un *passphrase* para generar las claves. Con esta instrucción obtenemos dos ficheros, la clave privada y la clave pública que es la que se distribuye entre las Raspberry-PIs para poder conectarse de forma segura.

Podemos crear abreviaciones para poder conectarnos a diferentes dispositivos, para ello creamos/editamos el fichero *.ssh/config* dentro de ese fichero podemos definir todos los dispositivos que queramos como podemos ver en la figura 3.3 los parámetros que debemos

introducir son 3: Debemos escribir el nombre con el que nos vamos a referir al dispositivo a partir de ese momento, debemos escribir la dirección IP del dispositivo en cuestión y el puerto que utilizaremos para la conexión SSH, si no definimos el puerto que va a usar SSH usará el que está definido por defecto que es el 22; hay otras opciones que se pueden añadir en este fichero pero estas son suficientes para nuestra implementación.

```
Host Name_shortened
        Hostname Host_IP
        User Host_user
        Port SSH_port (Default:22)
```

Figura 3.3: Formato archivo .ssh/config

Una vez se puede establecer y controlar las Raspberry-PIs mediante Secure Shell (SSH), se configuran los parámetros para el correcto funcionamiento de la red Ad-hoc que hemos creado anteriormente. Primero de todo tenemos que configurar las tarjetas de red para que reenvíen los paquetes que no van dirigidos para sí mismo, esta configuración se encuentra en el fichero */proc/sys/net/ipv4/ip_forward* en este fichero solo aparece un dígito binario, si aparece '0' significa que el nodo no reenvía los paquetes, por el contrario en el caso de aparecer un '1' si reenvía estos datos, como es lógico debemos poner un '1' en los nodos intermedios que son los que realizaran.

Por otra parte, tenemos que configurar las tablas de rutas de todas las Raspberries de manera estática, debido a que para cada Raspberry la configuración va a ser diferente, de hecho, dependiendo del nodo destino puede ser que no todas las Raspberries estén trabajando al mismo tiempo. El comando usado en Linux para poder añadir entradas a la tabla de rutas es: **route add *Destination* gw *Next-hop* dev *Interface***.

El parámetro *Destination* está claro que se trata del nodo destino que actuará como receptor en la transmisión que vamos a realizar.

El parámetro *Next-Hop* es el siguiente nodo al que se debe transmitir el paquete para alcanzar el destino. Para terminar, es necesario especificar la interfaz de salida por la que se enviará el paquete, como en este caso todas las transmisiones son Wi-fi, en una red Ad-hoc, la interfaz en cualquiera de los casos será *Wlan0*.

Para realizar la configuración de una manera más rápida se ha creado un script en *Python* con objeto de automatizar el proceso, lo podemos ver en la Figura 3.5

Ya tenemos la configuración necesaria para que los nodos puedan reenviar los paquetes, pero nosotros lo que queremos hacer es poder modificar esos paquetes en los nodos intermedios, este papel lo desempeña *Netfilter* como ya hemos comentado anteriormente en la Sección 2.7 esta herramienta permite la captura de paquetes, entrando más en detalle

nos permite capturar un paquete y poder manipularlo con un programa en el espacio de usuario.

Lo primero que debemos hacer es crear las reglas necesarias para poder filtrar los paquetes que necesitamos para esa tarea vamos a usar el comando:

```
sudo iptables -t Tabla donde se guarda la regla -A Hook
-s IP_origen -d IP_destino -p Protocolo --Opciones(si tiene)
-j Tarea a realizar por la regla --Opciones de la tarea
```

Figura 3.4: Formato reglas *Iptables*

Vamos a explicar un poco más en detalle cada uno de los parámetros de la orden que vemos en la figura 3.4:

- -t : Netfilter tiene diversas tablas donde poder almacenar las reglas, cada una con una prioridad diferente(filter,nat,mangle...) concretamente esta última es la que utilizaremos nosotros, ya que es la indicada para realizar modificaciones sobre los paquetes.
- -A : Este parámetro se usa cuando queremos dar opciones especificas a la regla, en nuestro caso la única opción de este tipo que vamos a utilizar es especificar a la herramienta el gancho donde queremos capturar los paquetes. Se capturan los paquetes justo cuando llegan al dispositivo, tras comprobar el *Checksum* IP, antes de que se determine hacia donde debe ser enrutado el paquete. Utilizaremos la opción -A PREROUTING.
- -s : Dirección IP origen del paquete.
- -d : Dirección IP destino del paquete.
- -p : Se puede especificar que tipo de protocolo se quiere filtrar, en nuestro caso no es necesario solo vamos a filtrar en función de la dirección origen y destino puesto que es el único tráfico en la red Ad-hoc.
- -j : Como explique en el apartado anterior Netfilter puede realizar diversas acciones sobre los paquetes, nosotros nos interesa enviar los paquetes al espacio de usuario por lo que en todos los casos la acción será *NFQUEUE*, junto con esta acción aparecen algunas opciones interesantes como son:
  - –queue-num : Puedes definir varias colas, cada una destinada a un tipo de paquetes o a un programa diferente.
  - –queue-bypass: Por defecto si hay establecida una regla con *NFQUEUE* y no se está ejecutando el programa de usuario que debe manejar ese paquete la

herramienta realiza un *DROP* sobre todos los paquetes de la regla, con esta opción si el programa no se está ejecutando el filtro deja pasar los paquetes, justo lo contrario.

Para que nuestros programas de usuario sean capaces de poder acceder a los paquetes de las colas de Iptables es necesario incluir dos librerías: *netfilter.h, libnetfilter_ queue.h*

Estas son todas las tareas que debemos realizar para la configuración de las Raspberries, pero realizar la configuración de todos estos parámetros uno a uno llevaría demasiado tiempo, por no hablar de que muchos de estos parámetros no son guardados cuando la placa se apaga.

Para simplificar este proceso vamos a hacer uso de un Script escrito en *Python* inspirado en el del trabajo[9] en ese script vamos a realizar todas las configuraciones necesarias:

- Establecer las tablas de rutas
- Activar el reenvío de paquetes en los nodos intermedios[2].
- Configurar las reglas necesarias para trabajar con los paquetes
- Establecer unos parámetros para el funcionamiento de las pantallas conectadas a las Raspberries.
- Resetear las trazas que recogen los datos.

Las Raspberries están numeradas de 1-20 y las IPs están asignadas de acuerdo a este número, empezando en la 192.168.3.11 en la red Ad-hoc y en la 192.168.2.11 en la red Ethernet. Con estos valores hemos creado un código el cual necesita como parámetros el vector de ruta, es decir los nodos que intervendrán en la comunicación, siendo el primer elemento del array el nodo origen y el último el nodo destino, por ejemplo $ruta = [1, 2, 3]$.

[2]Para facilitar este paso tenemos un fichero donde están habilitadas ya los reenvíos solo hace falta sustituirle por el actual en los nodos intermedios

```
def Asig (source, receiver, route):

  #Route forward
  for i in range(1,len(route)):
    os.system('ssh 2pi' + str(i) + ' \'sudo route add 192.168.3.'
    + str(receiver + 10) + ' gw 192.168.3.' + str(route[i] + 10)
    + ' dev wlan0\' ')
    os.system('ssh 2pi' + str(i) + ' \' sudo cp
    /home/pi/Raspberry/Tools/ip_forward_1
    /proc/sys/net/ipv4/ip_forward \' ')

  #Route backward
  r = range(2,len(route)+1)
  r=reversed(r)
  for i in r:
    os.system('ssh 2pi' + str(i) + ' \'sudo route add 192.168.3.'
    + str(source + 10) + ' gw 192.168.3.' + str((i-1) + 10) +
    ' dev wlan0\' ')
```

Figura 3.5: Función asignación de rutas

Para asegurarnos del correcto funcionamiento de todo antes de añadir las entradas a la tabla de rutas borramos todas las entradas existentes, después asignamos las rutas y, a posteriori, se configura la topología deseada. Los nodos intermedios se ponen a funcionar como *recoder* y el nodo receptor como *receiver* a la espera de información, figura 3.6.

```
source = route[0]
receiver = route[-1]

#Flush route table of devices
for i in route:
  os.system('ssh 2pi' + str(i) + ' \'sudo ip route flush dev wlan0\' ')
  os.system('ssh 2pi' + str(i) + ' \' sudo route add
  -net 192.168.3.0/24 dev wlan0\' ')
Asig(source, receiver, route)

for i in route:
  if (i != source and i != receiver):
    os.system('ssh 2pi' + str(i) + ' \'sudo iptables -t mangle -A
    PREROUTING -s 192.168.3.' + str(source + 10) + ' -d 192.168.3.'
    + str(receiver+10) + ' -j NFQUEUE \' ')

    os.system('ssh 2pi' + str(i) + ' \'cd Raspberry/RaspberryDemo;
    rm log.txt; rm traces/* ; export DISPLAY=:0.0; nohup sudo
    ./recoder >& log.txt\' &')
  elif (i == receiver):
    os.system('ssh 2pi' + str(i) + ' \'cd Raspberry/RaspberryDemo;
    rm log.txt; rm traces/* ; export DISPLAY=:0.0; nohup
    ./receiver >& log.txt\' &')
```

Figura 3.6: Script de inicialización

3.1.3. Implementación comunicación basada en NC entre nodos

Básicamente ha sido necesario implementar tres programas distintos, aunque bastante parecidos entre sí, *transmiter, recoder* y *receiver* toda esta parte ha sido elaborada con C++. A continuación, se detalla el trabajo realizado.

Se ha implementado un objeto que contiene todo lo necesario para poder realizar las acciones de codificación, recodificación y decodificación además de ser capaz de obtener las trazas resultantes, para el manejo de las trazas y exportarlas a un fichero hay otro objeto llamado *tracing* que es llamado por *decoder* por último también existe una cabecera propietaria (Figuras 3.8 y 3.7) que esta implementada con objeto de que todos los nodos puedan trabajar correctamente con los paquetes recibidos.

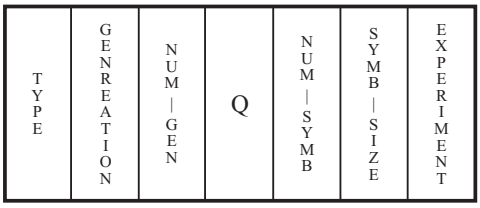

Figura 3.7: Cabecera inicial

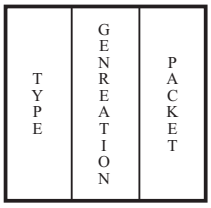

Figura 3.8: Cabecera de datos

- *Transmiter:* Lo primero que tiene que hacer este programa es leer los parámetros del fichero de configuración y crear un *socket* para poder enviar paquetes a través de la interfaz de red. Con los parámetros que ha leído del fichero de configuración crea un objeto *encoder*, que será el encargado de codificar los datos y enviar un paquete que inicia la transmisión siguiendo la estructura de la Figura 3.7 para que el resto de nodos configure sus opciones correctamente. El *encoder* requiere conocer la dirección del buffer donde se almacena la generación, así como su longitud. Por último, necesita otro puntero para poder almacenar los datos codificados listos para ser enviados. Una vez hecho todo esto el nodo transmite la cabecera de la figura 3.8 concatenada con el bloque de datos correspondiente.

- *Recoder:* Este nodo crea el objeto capaz de recodificar. A continuación se mantiene a la espera de los paquetes filtrados por Netfilter. El primer paquete que se recibe debe ser el que inicializa la comunicación, con los parámetros de configuración del tamaño de generación y cuerpo de Galois. En realidad el recodificador lo que hace es un proceso de lectura de los datos seguido de una escritura. En los nodos intermedios, utilizando la herramienta de IPtables, se pueden simular pérdidas de los enlaces, con objeto de analizar diferentes situaciones del medio inalámbrico. La captura de paquetes por Netfilter se realiza mediante una función de *callback*, no necesitamos crear *sockets* ni nada del estilo la herramienta se encarga de todo, dentro de esa función extraemos los datos del mensaje (*payload*), y les introducimos en *recod*, como

ya comenté en la Sección 2.4 el hecho de recodificar desde el principio aumenta mucho el número de dependencias lineales, por lo que antes de llegar al *Recod_ threeshold* el *recod* no recodifica los paquetes, solo les lee para tener más información disponible, a partir de ese punto empieza a recodificar. Tenemos que tener en cuenta el cálculo de nuevo de la cabecera de UDP, puesto que Netfilter nos entrega el datagrama IP completo.

- *Receiver:* Crea un *socket* que escucha en la dirección deseada, también habilitamos las pantallas para funcionar. Esperamos recibir el primer paquete para poder configurar el *decoder* y el nodo se pone a la espera de recibir la información.

Mencionar que tanto en los nodos intermedios como en los nodos destino, se establece un *Time_ out*, de forma que si transcurre dicho tiempo sin recibir ningún paquete el sistema se reinicia a la espera de un nuevo paquete de inicio de la sesión. Además que todos los nodos durante su ejecución están recogiendo trazas con datos interesantes como número de dependencias lineales, tiempo de transmisión, throughput...

3.2. Entorno de simulación

Con objeto de ampliar los resultados obtenidos mediante el banco de pruebas se utiliza el simulador NS-3 [16], un potente simulador para entornos de red.

NS-3 es un simulador de red para eventos discretos, está destinado para usos educativos y de investigación, se trata de un software con licencia GNU por lo que es libre y gratuito. El núcleo y los módulos están escritos en C++, aunque a día de hoy muchas de las APIs de esta herramienta también se pueden encontrar en Python. La descripción de este simulador está basada en el trabajo [17].

3.2.1. Características básicas

Este simulador comenzó a desarrollarse en el año 1989 en los laboratorios (Lawrane Berkely National Laboratory (LBNL)), la primera versión de este simulador (ns-1) apareció en torno a 1995. Más tarde con la colaboración de Sun Microsystems y UC berkley surgió *ns-2*. Por último, en 2004 apareció *ns-3* gracias al trabajo de la Universidad de Washington. Hay que decir que *ns-3* no es compatible con sus antecesores, debido a la enorme carga de soporte necesaria para ello. La última versión estable de esta herramienta es *ns-3.28* sin embargo se va a trabajar con la versión *ns-3.20* el motivo es poder usar los módulos desarrollados por el Grupo de Ingeniería Telemática de la Universidad de Cantabria, ya que el simulador no cuenta con ningún módulo capaz de trabajar con NC.

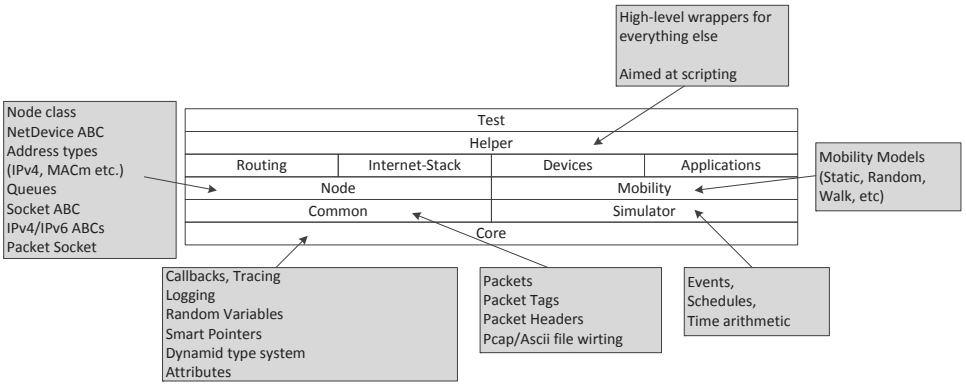

Figura 3.9: Arquitectura del software de NS-3

Este programa ofrece una gran flexibilidad y modularidad permitiendo el uso solo de las herramientas necesarias, el problema de esto es que resulta un poco complejo el manejo del simulador, sobre todo debido a la compleja arquitectura interna, como se puede ver en la Figura 3.9. El uso del simulador es complejo, en parte por la gran cantidad de opciones que se pueden configurar. En este trabajo se utilizará Wi-fi para la conexión entre los nodos, sin embargo, la herramienta soporta otras tecnologías como CSMA, LTE, WiMax.

Para simplificar esta tarea se va a hacer uso de un módulo desarrollado por el Grupo de Ingeniería Telemática, este se encarga de definir la topología deseada utilizando ficheros de configuración. El código fuente se encuentra en la carpeta *src*, aunque no se va a entrar en detalle de su contenido es de vital importancia tener claro la estructura que sigue el simulador para su correcto uso.

Nodo	1	2	3	4
1	100	80	0	15
2	80	100	0	100
3	0	0	100	60
4	15	100	60	100

Tabla 3.2: FER de los enlaces

3.2.2. Creación de escenarios

El programa capaz de crear los escenarios facilitando el uso del simulador, necesita cuatro archivos para poder configurar los escenarios de manera correcta: Distribución física de los nodos en el espacio, características del canal de comunicación, configuración de las rutas y parámetros propios de la simulación. Se explicarán en más detalle a continuación.

- Distribución física de los nodos en el espacio: Hay un fichero donde tenemos una matriz de x filas(una por cada nodo que posea la red) y de seis columnas, la primera de ellas sirve para enumerar al nodo en cuestión, las tres siguientes indican la posición del nodo en el espacio de acuerdo a los 3 ejes x, y, z, y las dos últimas (TX y RX) especifican que nodos son los que transmiten y a que nodos. En la Tabla 3.1 podemos ver un ejemplo de implementación, se ilustra la configuración en la Figura 3.10.

Nodo	X	Y	Z	TX	RX
1	0	0	0	4	0
2	15	0	0	0	0
3	30	0	0	0	0
4	15	15	0	0	1

Tabla 3.1: Matriz de la distribución física de los nodos en el espacio

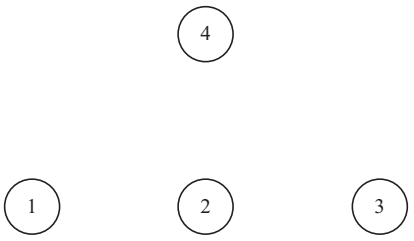

Figura 3.10: Ejemplo Topología NS-3

- Características del canal de comunicación: Al tratarse del medio inalámbricos la forma más sencilla de caracterizarle es a través de la Frame Error Rate (FER) de esta forma

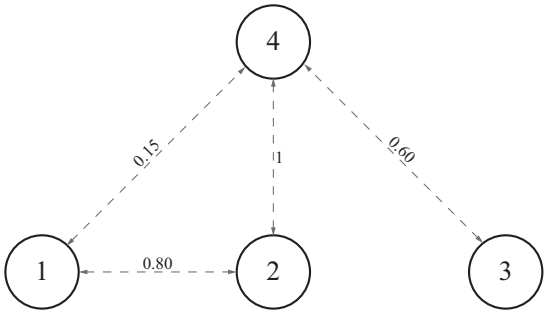

Figura 3.11: Ejemplo FER NS-3

en este archivo podemos determinar la FER que deseemos para un enlace concreto siendo '0' un canal ideal sin perdidas y un '100' un enlace inexistente, cualquier valor intermedio es válido. Estos valores los vamos a introducir en una matriz cuadrada simétrica, puesto que el enlace se comporta de igual manera en ambos sentidos. Un fichero de caracterización de la topología que se muestra en la Figura 3.11 se corresponde con la Tabla 3.2.

- Configuración de las rutas: Para nuestro trabajo, al igual que para el banco de pruebas, vamos a usar encaminamiento estático de los paquetes para este fichero tenemos que definir para cada nodo, todas las rutas con el resto de nodos, es decir, el siguiente salto la interfaz y la métrica si fuese necesario (en nuestro caso la métrica para todas las entradas de la tabla es '0') se puede visualizar mejor en la Tabla 3.3.

Nodo	Destination	Next_Hop	Interface	Metric
1	2	2	1	0
1	3	2	1	0
1	4	4	1	0
2	1	1	1	0
2	3	4	1	0
2	4	4	1	0
3	1	4	1	0
3	2	4	1	0
3	4	4	1	0
4	1	1	1	0
4	2	2	1	0
4	3	3	1	0

Tabla 3.3: Tabla de rutas del escenario

- Parámetros propios de la simulación: El último fichero que queda de configurar es el que determina todos los parámetros de la comunicación como el protocolo de transporte, la tasa de datos, el número de retransmisiones Wi-fi, ... Además de esto también es el encargado de todos los parámetros referentes a NC como el número de símbolos, si se realiza recoding o no, el umbral (en caso de que haya recoding). Por último, se pueden configurar también los datos que queremos extraer de las simulaciones eligiendo que trazas queremos obtener(Nivel aplicación, transporte, físico, ...). Podemos ver un ejemplo de este fichero en la Figura 3.12

```
[SCENARIO]
RUN=40
RUN_OFFSET=0
NUM_PACKETS=10000
PACKET_LENGTH=1300
SCENARIO_DESCRIPTION=three-nodes-scenario.conf
CHANNEL_CONFIGURATION=three-nodes-channel.conf
STATIC_ROUTING_TABLE=three-nodes-static-routing.conf

[STACK]
TRANSPORT_PROTOCOL=UDP

[APPLICATION]
RATE=50Mbps

[NETWORK_CODING]
ENABLED=0
PROTOCOL=ACKs
BUFFER_TIMEOUT=300
Q=1
K=2
RECODING=1
OVERHEARING=0
SYSTEMATIC=0
START_RECODING=0
CCODING_SCHEME=RLNC
MULTICAST=0

[TUNABLE]
FUNCTION=CONSTANT
INITIAL=3

[WIFI]
TX_NUMBER=4
DATA_RATE=54

[OUTPUT]
FILE_NAME=80211g
APPLICATION_LEVEL_SHORT_TRACING=1
TCP_LEVEL_SHORT_TRACING=1
NETWORK_CODING_SHORT_TRACING=1
PHY_WIFI_SHORT_TRACING=1
```

Figura 3.12: Archivo de configuración NS-3

4

Resultados

Esta parte se va a dedicar a detallar el procedimiento que se ha llevado a cabo para obtener las medidas oportunas, así como para exponer los resultados y los análisis de los mismos.

4.1. Conceptos básicos

Analicemos un poco los parámetros que van a condicionar los resultados obtenidos en el Sección 4.3

- **Parámetros NC**: Vamos a explicar los diferentes valores que condicionan el rendimiento del NC en función de la situación de la comunicación:
  - **Throughput**: Es un valor que determina la información "neta" recibida a través de un canal de comunicación. Es el cociente entre los datos útiles y el tiempo utilizado para su transmisión, expresa la velocidad de transmisión efectiva de información, esta velocidad siempre va a ser inferior a la capacidad física del enlace, debido al overhead, fallos en la transmisión, etc. Este valor se puede obtener a través de la Expresión 4.1.

$$Throughput = \frac{\text{Bytes de Información}}{\text{Tiempo de transmisión}} \tag{4.1}$$

- **Número de símbolos (K)**: Este valor indica el número de símbolos que posee cada generación, es decir, el número de paquetes que se combinan conjuntamente para crear una generación. Para poder decodificar correctamente la generación son necesarios 'K' paquetes linealmente independientes.
- **Tamaño del símbolo**: Numero de bytes que contiene cada uno de los K símbolos que sirven para realizar la codificación, este valor debe estar acotado entre un rango, debido a que el MTU máximo es de 1500 Bytes aunque hay que descontar los tamaños de las cabeceras (20 B IP + 8 B UDP),
- **q**: El valor 'q' determina el tamaño del cuerpo de *Galois* que será utilizado GF(2^q). Los coeficientes de codificación toman valores aleatorios entre 0 a 2^{q-1}. Hay que tener en cuenta que cuanto mayor sea el valor de q mayor será la longitud del vector de coeficientes, ya que el número de bits para codificar cada uno de los elementos aumenta conforme aumenta q. Por otra parte, el número de paquetes linealmente independientes también se ve afectado al variar este valor.
- **Umbral de *recoding***: Como ya se adelantaba en apartados anteriores este valor determina en gran medida el número de paquetes linealmente independientes que vamos a recibir, no hay una forma cerrada de calcular este valor, se va a presentar diferentes resultados en función de este valor para tratar de aproximarnos al valor óptimo.

- **Parámetros de la red**: Se comentan algunos parámetros característicos de las redes inalámbricas, algunos de ellos no van a poder ser modificados debido a la escasez de personalización de las tarjetas de red de las Raspberries.

  - **Red ad-hoc**: Tenemos configurada la red de forma que trabaje en modo ad-hoc a una tasa de 54 Mbps, esto implica algunos problemas debidos a la capacidad de procesamiento de los dispositivos, los bufferes en recepción pueden llegar a colapsarse impidiendo la recepción de más paquetes esto puede desembocar en grandes pérdidas lo que consecuentemente produce bajadas drásticas del rendimiento por la pérdida de generaciones. Los estándares Wi-fi tienen un mecanismo para paliar estas bajadas de la calidad del enlace, se denomina *Adaptative Rate Selection*, la transmisión cambia la constelación utilizada o la tasa de símbolos para contrarrestar el aumento de perdidas, esto genera una baja de la tasa binaria pudiendo llegar hasta los 2 Mbps.
  - **FER**: Este valor es uno de los más determinantes a la hora de decidir en qué escenarios es interesante el uso de NC, ya que los protocolos tradicionales sufren caídas de rendimiento mucho mayores cuando empeora el estado del enlace.
  - **Retransmisiones a nivel físico**: El estándar Wi-fi 802.11, que es el que se encarga de especificar la capa física y Media Access Control (MAC), establece un numero de retransmisiones con objeto de conseguir el envío correcto de la

información. Esta acción está pensada para tratar de garantizar la llegada de información pero en nuestro caso no es útil, ya que lo único que hace es congestionar la red, las tarjetas de red de los nodos no se pueden configurar para poder quitar este parámetro por lo que no tenemos una medida real del rendimiento ni con NC ni con TCP, con ausencia de retransmisiones TCP se vería notablemente perjudicado, cosa que nuestra implementación sería capaz de solventar sin grandes inconvenientes.

4.2. Método de medida

Vamos a presentar resultados en diversos escenarios todos ellos basados en el esquema presentado en la Figura 3.2, contrastados mediante el simulador ns-3 (Sección 3.2) y con el modelo teórico, variando el número de nodos intermedios en la topología, otro parámetro que también probaremos a modificar es la FER del enlace.

Usaremos las trazas propias de nuestra implementación para medir el rendimiento de NC, probaremos tanto RLNC sin recodificación, como con recodificación, variando el umbral de recodificación, también obtendremos trazas del simulador que nos proporcionan los mismos valores.

Debido a las variaciones que sufre el medio inalámbrico vamos a tomar 100 medidas para cada uno de los escenarios y para cada posible valor del número de símbolos (K), con objeto de obtener datos fiables del banco de pruebas. Los valores que aparecen en las gráficas son los valores medios obtenidos de estas medidas, también se representará en las gráficas el intervalo de confianza de los resultados, en nuestro caso usamos un intervalo del 95 % [1], para calcular este intervalo usamos la Expresión 4.2 No vamos a analizar los diferentes valores de 'q', utilizaremos siempre q=1.

$$Confianza = 2.2281 \times \frac{\sqrt{\text{Varianza de los resultados}}}{\sqrt{\text{Número de resultados}}} \tag{4.2}$$

Para todas las simulaciones vamos a usar un tamaño de símbolo de 1400 B, podríamos usar un valor todavía algo superior ya que la Maximum transmission unit (MTU) es de 1500 B (aunque hay que contar las cabeceras). La especificación de nivel físico y acceso al medio que hemos utilizado, tanto para el TestBed como para el simulador, es IEEE 802.11g que puede alcanzar velocidades de enlace de 54 Mbps.

[1] Para obtener este intervalo de confianza multiplicaremos por un factor $t = 2.2281$

4.3. Análisis de los resultados

En esta sección vamos a presentar gráficas que representan diferentes formas de analizar el rendimiento de usar *recoding* sobre redes inalámbricas. Los valores por analizar van a ser los siguientes:

- Sobrecarga debido a las dependencias lineales en función del tamaño de la generación K, analizaremos estos resultados tanto sobre la plataforma de experimentación como en el simulador, ambas gráficas se contrastaran con los resultados del modelo teórico.
- *Umbral* óptimo, este valor es dependiente de la FER de los enlaces.
- Throughput de los diferentes protocolos de transporte (TCP y UDP) así como del uso de *NC* con y sin *recoding*.
- Sobrecarga debido a las dependencias lineales en función del número de saltos, analizaremos estos resultados tanto sobre la plataforma de experimentación como en el simulador, ambas gráficas se contrastarán con los resultados del modelo teórico.
- Nº de transmisiones para NC sin recoding, y NC con recoding usando el umbral óptimo.

Para la obtención de los resultados, salvo para el caso de dependencias lineales en función del número de nodos, se ha utilizado un escenario basado en 3 nodos como el de la figura 2.9

4.3.1. Sobrecarga debido a paquetes linealmente dependientes, en función de K

En este apartado se van a presentar dos gráficas la primera de ellas elaborada a partir de los datos de la plataforma de experimentación, mientras que la segunda se ha obtenido mediante el simulador. Se realizan 100 iteraciones para cada uno de los valores de configuración, en cada experimento se transmiten 100 generaciones. Los resultados muestran las dependencias lineales de media en cada generación. Además, también representamos el intervalo de confianza correspondiente al 95 %.

En la Figura 4.1 vemos como a medida que aumenta el valor de K aumenta el número de dependencias lineales que se reciben, sin embargo, es fácil de apreciar que la sobrecarga relativa disminuye con el tamaño de la generación.También se valida el modelo teórico, Ecuación 2.8, como se puede ver en la Figura 4.1 los resultados obtenidos mediante experimentación coinciden con los obtenidos con el modelo teórico.

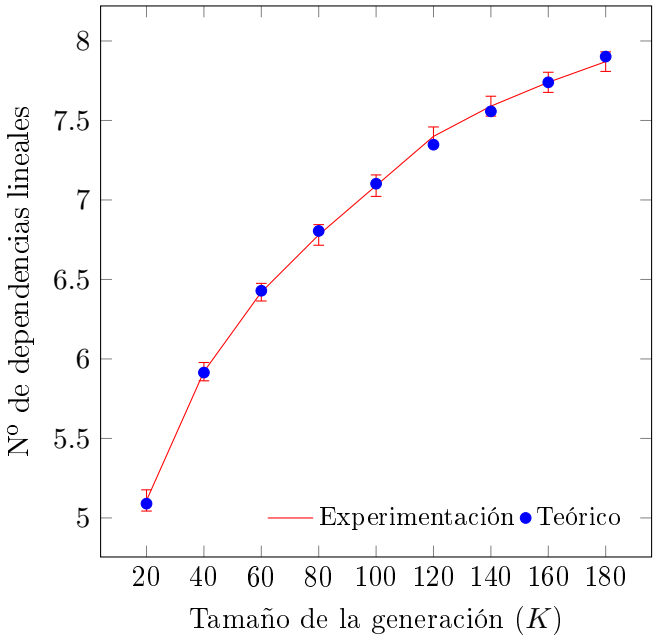

Figura 4.1: Número de dependencias lineales en función de K(Raspberry)

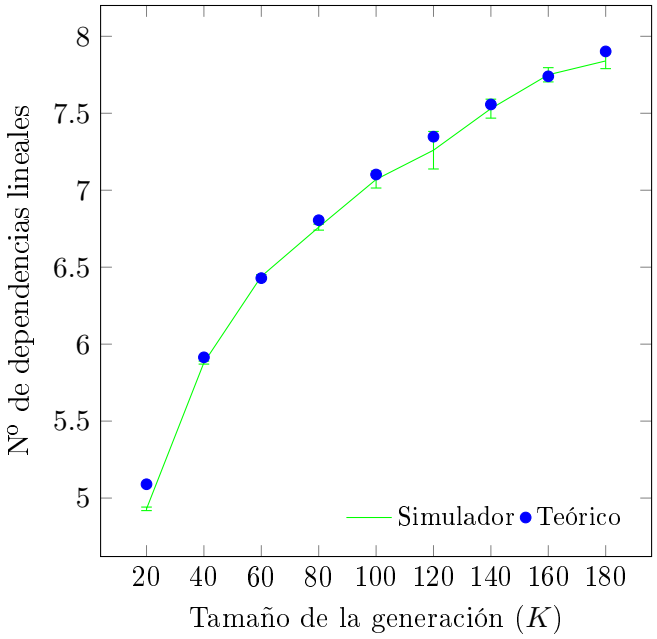

Figura 4.2: Número de dependencias lineales en función de K(Simulador)

En la Figura 4.2 podemos contrastar los resultados obtenidos mediante experimentación con los valores obtenidos a través del simulador *ns-3*. Como era de esperar los resultados son muy similares entre sí.

4.3.2. Umbral óptimo de *recoding*

Como ya se adelantaba en el Apartado 2.3 determinar el umbral óptimo no es un método simple ya que no se conoce ninguna expresión que sea capaz de determinarlo. En el trabajo [2] se puede ver una tabla con los valores de umbral óptimo para un escenario de 3 nodos. Todos estos valores han sido obtenidos a partir del modelo analítico probando todos los posibles valores del umbral, desde 0 hasta $K + 1$ escogiendo el que ofrecía un menor número de transmisiones, para unas determinadas características del canal, ya que como he comentado no se conoce otra forma de hallar estos valores.

$p_1 \backslash p_2$	*0.0*	*0.1*	*0.2*	*0.3*	*0.4*
0.0	101	101	101	101	101
0.1	101	29	16	11	9
0.2	101	19	11	8	6
0.3	101	15	8	6	5
0.4	101	22	7	5	4

Tabla 4.1: Umbral óptimo para transmisiones con $K = 100$

Se puede apreciar que en enlaces ideales sin pérdida de paquetes no aporta ninguna ventaja el uso de *recoding*, de hecho, en enlaces ideales la mejor forma de transmitir la información sería usar UDP. A medida que empeoran las características del enlace antes es recomendable recodificar, no se recibirán tantos paquetes linealmente dependientes ya que las pérdidas de información son mayores.

4.3.3. Throughput

El escenario sobre el que hemos realizado estas medidas es un entorno de 3 nodos, consideramos los enlaces entre nodos ideales, es decir la FER de todos los enlaces es igual a '0'. En el eje horizontal se varía el tamaño de la generación, mientras que en el eje vertical se muestra el Throughput de los diferentes modos de transmisión analizados. Los diferentes valores de K que se representan son: 2, 4, 8, 16, 32, 64, 128, 255.

Para los resultados de NC, al igual que en la gráfica anterior se han realizado 100 iteraciones de 100 generaciones cada una.

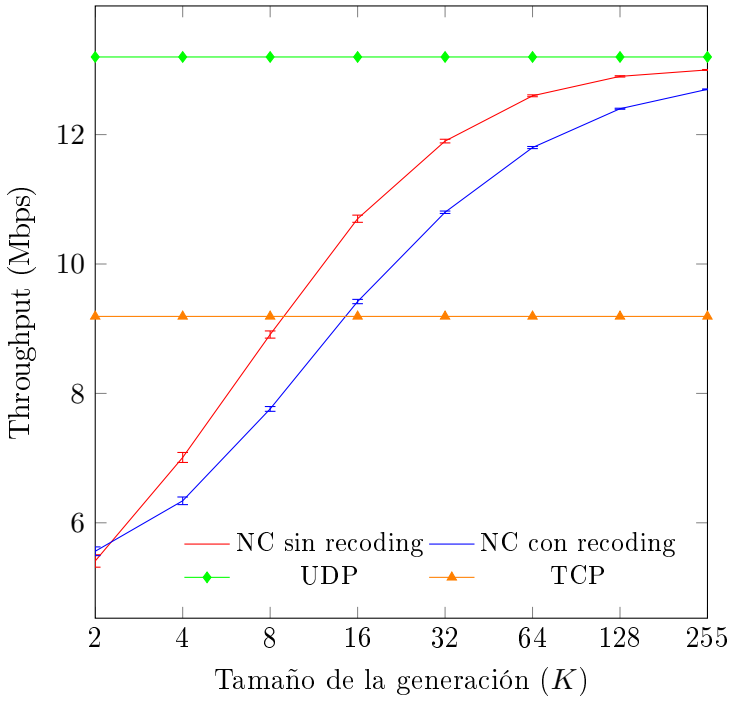

Figura 4.3: Throughput 802.11g de las diferentes formas de envío

En la Figura 4.3 podemos ver, como era previsible que el protocolo que mejor rendimiento ofrece es UDP, aunque esto es principalmente debido a que es un protocolo best-effort que no garantiza la llegada de los datos, ni la entrega ordenada de los mismos. Vemos también que las 2 variantes que usan NC están por encima de TCP en términos de rendimiento, tener en cuenta que estos valores han sido calculados suprimiendo las retransmisiones a nivel 802.11 para el uso de NC, mientras que TCP mantiene el número de retransmisiones a 4. Por último, apreciar que el uso de *recoding* en este escenario no aporta ninguna ventaja frente al esquema "clásico" esto es debido a que empezamos a codificar desde el primer momento, lo que provoca una gran cantidad de transmisiones innecesarias.

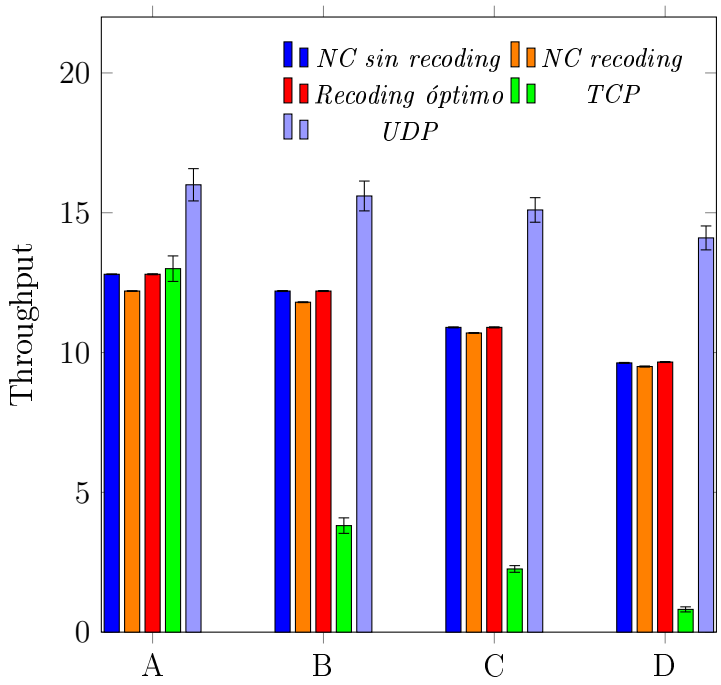

Figura 4.4: Throughput de las diferentes variantes NC usadas

	A	B	C	D
p_1	0.00	0.00	0.05	0.05
p_2	0.00	0.05	0.10	0.20

Tabla 4.2: Parametros de la figura 4.4

En la Figura 4.4 podemos ver la clara mejora que obtenemos al utilizar el umbral óptimo de *recoding*, valor dependiente de las condiciones del canal como hablaremos en el Apartado 4.3.2, el rendimiento sería prácticamente el mismo que en el caso de usar NC sin *recoding* sin embargo el uso de los nodos intermedios para codificar la información aporta otras ventajas que se observaran posteriormente.

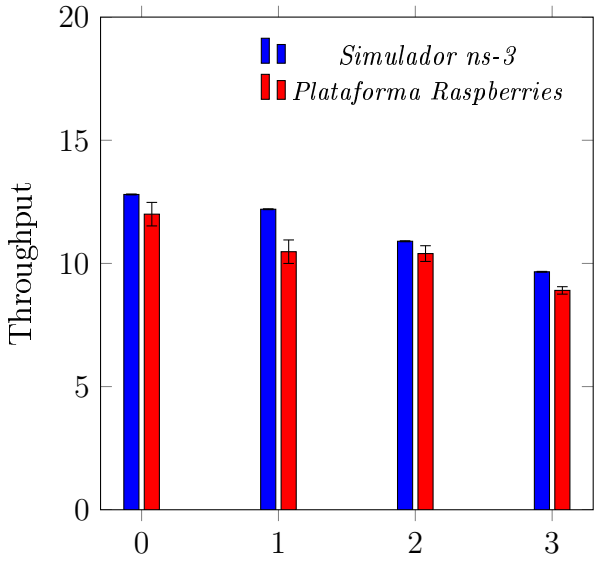

Figura 4.5: Throughput *Raspberry vs ns-3*

Observar en la Figura 4.5 que el rendimiento obtenido en la plataforma es inferior al que hemos obtenido mediante *ns-3* esto es en parte debido a que el despliegue real se tiene que enfrentar a cambios constantes en el medio radio, por no hablar de las interferencias que existen en el laboratorio de Telemática causadas por el resto de dispositivos. También se observa que los intervalos de confianza que ofrece la plataforma son mucho peores que los que entrega el simulador, esto hasta cierto punto es normal puesto que estamos haciendo uso de una plataforma real donde las condiciones del canal sufren de mayor variabilidad.

4.3.4. Sobrecarga debido a paquetes linealmente dependientes, en función del número de saltos

Ahora se representa el número de dependencias lineales en función del número de saltos realizados, con uno dos o tres saltos, aparecen valores tanto del simulador como de las Raspberries sin utilizar *recoding*, por último, se representa los mismos escenarios usando *recoding* con un umbral óptimo. Para obtener estos resultados se ha fijado el tamaño de la generación (K) a 100.

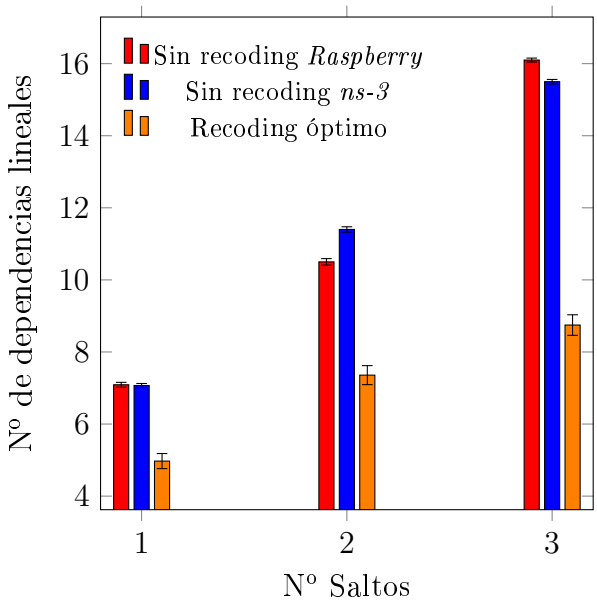

Figura 4.6: Número de dependencias lineales en función de los saltos

En la Figura 4.6 se puede observar el número de dependencias lineales, vemos como el número de paquetes linealmente dependiente crece con el número de saltos, los nodos intermedios no poseen la misma información que el nodo origen, esto provoca que la información que envían tenga más probabilidad de ser redundante, al aumentar el número de nodos este efecto es más acusado. Vemos que el uso de *recoding* en este tipo de topologías ofrece grandes ventajas

4.3.5. Nº de transmisiones necesarias

Ya hemos hecho una comparativa del rendimiento que ofrece el uso o no de *Recoding* en términos de velocidad, ahora vamos a analizar el número de transmisiones que son necesarias para enviar una generación de 100 símbolos (K = 100).

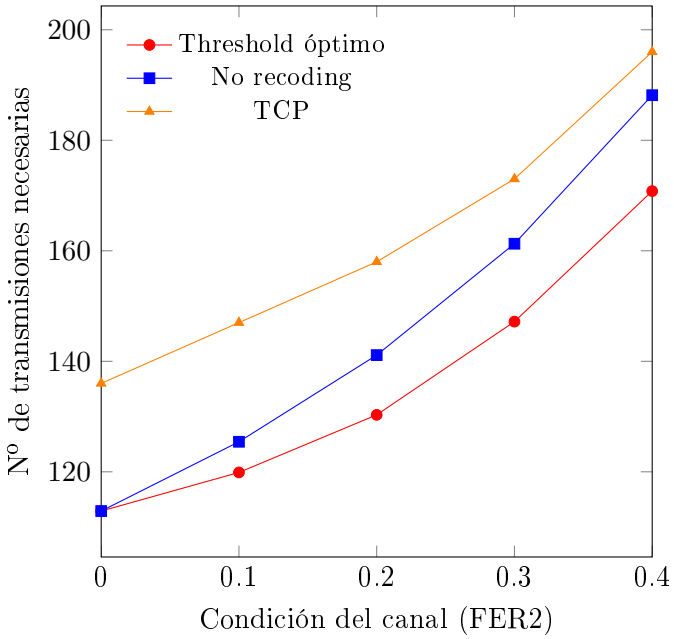

Figura 4.7: Número de transmisiones para una FER1 del 10 %

Como se aprecia en la Figura 4.7 el número de transmisiones necesarias para poder enviar un paquete con el uso del umbral óptimo es menor que sin el uso de recoding, esta diferencia va aumentando a medida que se empeoran las condiciones del canal. También apreciamos que en todo momento el número de transmisiones necesarias de TCP, cuando el canal presenta perdidas, es en todos los casos mayor al número de transmisiones NC

5

Conclusiones y líneas futuras

Esta última parte del trabajo resume y analiza todo lo expuesto en los apartados anteriores, poniendo especial atención a los resultados obtenidos en el apartado 4. Por otra parte, se hará una pequeña estimación del posible futuro del uso de esta técnica.

5.1. Conclusiones

Debido al cambio que se está produciendo en las telecomunicaciones desde hace unos años son cada vez más los dispositivos que demandan conectividad inalámbrica. A esto también tenemos que sumar la extensión geográfica en aumento, además del aumento de la capacidad que requieren los nuevos servicios como la transmisión de vídeo.

Hoy en día las redes inalámbricas es una de las formas de conexión más utilizadas y una de las soluciones más interesantes para ofrecer amplias áreas de conectividad son las redes malladas. Estas redes presentan algunos inconvenientes entre los que destaca las malas condiciones del medio radio, el esquema de comunicación tradicional, TCP, es conocido por no ser capaz de ofrecer un alto rendimiento sobre este tipo de redes. Para solventar estos inconvenientes se propone el uso de esquemas de codificación como puede ser el NC.

La mayoría de los estudios en este campo trabajan sobre modelos teóricos o en entorno de simulación, y son pocos los trabajos que han analizado el funcionamiento de NC sobre

entornos reales donde no se prueba el funcionamiento real de un sistema como este. En este trabajo se realizan pruebas en un banco de pruebas compuesto por dispositivos de bajo coste como son Raspberry-PIs. Concretamente se exploran las posibilidades que ofrece el uso de *recoding* sobre un entorno multi-salto.

Se han probado diferentes implementaciones variando algunos de los posibles parámetros de este esquema de codificación. Hemos probado diferentes valores para el tamaño de las generaciones K, se han probado diferentes escenarios variando el número de nodos, y también se han probado diferentes valores del umbral a partir del cual los nodos intermedios empiezan a formar parte activa en la comunicación.

Se observa como a medida que se aumenta el tamaño de la generación la velocidad de transmisión crece. Aunque este valor no puede configurarse todo lo grande que desearíamos debido a que como ya sabemos este esquema requiere una importante capacidad de computo, capacidad que se ve incrementada a medida que crece el tamaño de la generación.

También se han recogido resultados del número de paquetes linealmente dependientes que recibimos, el número de dependencias lineales aumenta a medida que se incrementa K, aunque proporcionalmente hablando ($\frac{\text{Dependencias}}{K}$) este número disminuye. Este número también crece a medida que aumenta el número de nodos intermedios, especialmente en el caso de realizar *recoding* desde el principio de la transmisión.

Con idea de poder reducir el número de paquetes linealmente dependientes se ha tratado de obtener el umbral óptimo para comenzar a recodificar los paquetes (obtenido de [2]), este valor solo puede obtenerse a partir del modelo teórico descrito, probando todos los posibles valores.

Una vez hallado este parámetro comparamos el número de transmisiones necesarias tanto para el uso de NC sin *recoding* como para el uso de *recoding* en el punto óptimo. El esquema propuesto ofrece un menor número de transmisiones y un mayor throughput que en esquemas donde el nodo intermedio nunca o siempre recodifica.

Hay que comentar que los resultados obtenidos mediante la experimentación con Raspberries PI serían mejores en el caso de que estuviesen deshabilitadas las retransmisiones 802.11, ya que la personalización que ofrecen las tarjetas de red de estos dispositivos no es suficiente para poder deshabilitarlas.

Por último, reflejar que parte de los resultados obtenidos en este trabajo así como el banco de experimentación implementado en este trabajo de fin de grado han sido usados en el *paper* **'To Recode or Not to Recode: Optimizing RLNC Recoding and Performance Evaluation over a COTS Platform** publicado en la conferencia *European Wireless 2018*.

5.2. Líneas futuras

Aunque ya hemos visto que el uso de *recoding* en situaciones adversas del canal es beneficioso, todavía existen numerosos aspectos por resolver sobre esta técnica.

Quizás la cuestión más importante que queda pendiente es establecer una política óptima en los nodos intermedios para que sean capaces de determinar en qué situaciones deben usar *recoding* y en cuales no, como ya hemos visto el uso de esta técnica no es siempre beneficioso para la transmisión, solo en los casos en los que se producen ciertas perdidas debido al canal, en función de estas pérdidas se establece el punto donde los nodos intermedios deben recodificar los paquetes. No existe una forma cerrada de obtener este valor y mucho menos una manera de determinar a priori si un nodo debe recodificar o no.

Por otra parte, se deben realizar pruebas sobre topologías más complejas, nosotros hemos realizado únicamente experimentos con escenarios multi-salto con un máximo de 5 nodos. También sería útil analizar el rendimiento de esta técnica utilizando otras métricas, como puede ser el consumo de energía, un aspecto cada vez más importante en los dispositivos inalámbricos debido a que determinan su autonomía.

Para terminar, es interesante la idea de combinar técnicas de NC con protocolos multi-camino para analizar su rendimiento frente a los métodos utilizados hoy en día.

Bibliografía

[1] M. Zorzi, A. Chockalingam, and R.R. Rao. Throughput analysis of tcp on channels with memory. *Selected Areas in Communications, IEEE Journal on*, 18(7):1289–1300, July 2000.

[2] Alfonso Fernández Pablo Garrido, Ramon Agüero. To recode or not to recode: Optimizing rlnc recoding and performance evaluation over a cots platform. 2018.

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

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

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

[6] 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.

[7] 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.

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

[9] Fátima Fernández Pérez et al. Despliegue de una plataforma para análisis de técnicas de network coding utilizando raspberry pi. 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] John G Kemeny, J Laurie Snell, and A Knapp. *Markov Chains*. Springer-Verlag, New York, 1976.

[12] 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.

[13] Steinwurf. Kodo documentation, http://docs.steinwurf.com.

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

[15] Netfilter documentation, https://netfilter.org/.

[16] Ns-3 documentation, https://www.nsnam.org.

[17] P. Garrido. Algoritmo para determinar la posibilidad de aplicar técnicas de codificación de red inter-flujo sobre redes malladas inalámbricas, 7 2014.

Lista de acrónimos

WMN Wireless Mesh Network

NC Network Coding

FER Frame Error Rate

UDP User Datagram Protocol

TCP Transport Control Protocol

SSH Secure Shell

SFTP Secure File Transfer Protocol

RLNC Random Linear Network Coding

TSNC Tunable Sparse Network Coding

MTU Maximum transmission unit

AP Access Point

IoT Internet of the Things

VoD Video on Demand

ACK Acknowledgement

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

IP Internet Protocol

API Application Programming Interface

OSI Open System Interconnection

LBNL Lawrane Berkely National Laboratory

MAC Media Access Control