Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Proyecto Fin de Carrera
Ingeniería en Informática

Implementación de una pasarela entre el protocolo RT-WMP y TCP/IP

Rubén Durán Balda

Director: Danilo Tardioli
Ponente: José Luis Villarroel Salcedo

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Curso 2011/2012
Noviembre de 2011

A mis padres y mi hermana.

Índice general

Capítulo 1
Introducción

El protocolo *RT-WMP* [14] (Real-Time Wireless Multi-hop Protocol), desarrollado dentro del Grupo de Robótica, Percepción y Tiempo Real, es un protocolo de tiempo real con multitud de aplicaciones de distinta naturaleza. Su aplicación más común es la de conectar a pequeños equipos de robots móviles (véase por ejemplo [6]) u ofrecer conexión de voz a nodos móviles en entornos complicados [9]. Sin embargo, las redes construidas con este protocolo se encuentran aisladas del mundo exterior. Esto implica que no es posible tener acceso a los nodos que la constituyen desde fuera ni acceder a redes externa (por ejemplo Internet) desde dentro de la red.

Con este proyecto se intenta superar estas limitaciones, permitiendo el uso de aplicaciones convencionales sobre estas redes y su interconexión con redes basadas en *IP*. Además, se permitirá que las aplicaciones funcionando sobre redes *RT-WMP* hagan uso de sus características, tales como la asignación de prioridades estáticas o la gestión de la calidad de servicio en las comunicaciones.

1.1 Contexto

Este proyecto se ha desarrollado en el Laboratorio de Robótica del Instituto de Investigación en Ingeniería de Aragón(I3A), dentro del Grupo de Robótica, Percepción y Tiempo Real del Departamento de Informática e Ingeniería de Sistemas (DIIS) y más concretamente en el marco del proyecto *TESSEO* (Sistemas Multi-Robot en Aplicaciones de Servicio y Seguridad - TEams of robots for Service and Security missiOns), que se encarga de la investigación de técnicas para que sistemas multi-robot actúen coordinadamente en escenarios realistas.

En este grupo, se emplea el protocolo *RT-WMP* para múltiples aplicaciones, algunas de las cuales son:

- Desplegar grupos de robots en misiones de reconocimiento y rescate, permitiéndoles establecer una comunicación en tiempo real entre ellos y mantener el contacto con el puesto de control.
- Ofrecer cobertura en situaciones y lugares donde no se disponga de las redes de comunicación convencionales, como puede ser el interior de un túnel o de una cueva, por medio del despliegue de una serie de nodos comunicados por este protocolo.

Algunos ejemplos de los trabajos realizados por este grupo son: Sistema multi-robot en localización e identificación de vehículos [6], Enforcing Network Connectivity in Robot Team Missions [13] y Real-Time Wireless Multi-Hop protocol in Underground Voice Communication [9], entre muchos otros.

RT-WMP es un protocolo que soporta tráfico de tiempo real estricto para redes móviles ad hoc (Mobile ad hoc network, MANET) y que funciona sobre dispositivos que utilizan el protocolo 802.11. Emplea un esquema de paso de testigo para garantizar tiempos límite de transmisión y está completamente descentralizado. El protocolo soporta prioridad estática de mensajes y cambios frecuentes en la topología de la red gracias al intercambio entre los nodos de una matriz de adyacencia con información de la calidad del enlace entre estos. Un ejemplo de respuesta a un cambio de la topología puede observarse en la figura 1.1.

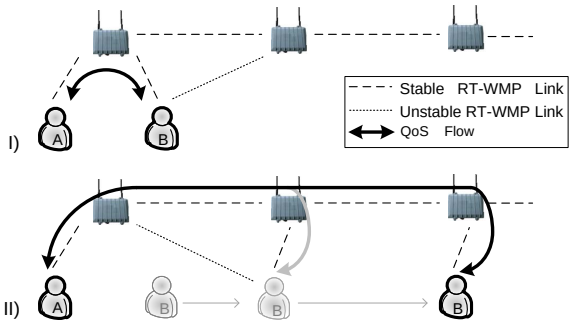


Figura 1.1: **Cambio de nodo en función de la calidad del enlace**

El protocolo *RT-WMP* se explicará en el siguiente apartado de esta memoria.

1.2 Objetivos

Si bien el protocolo *RT-WMP* cumple con su objetivo, también tiene ciertas limitaciones que se desean superar. Estas limitaciones radican en el hecho de que una red comunicada por este protocolo se encuentra aislada del exterior y en que para las comunicaciones dentro de esta no pueden emplearse programas de comunicación convencionales. Esto es debido a que estos están diseñados generalmente para comunicaciones empleando el protocolo de Internet (*IP*) y más específicamente *TCP/IP*, mientras que para las comunicaciones dentro de la red se deben emplear programas diseñados específicamente para funcionar sobre *RT-WMP*.

En la configuración disponible hasta el momento el *RT-WMP* es una libería que solo pueden usar procesos que se ejecutan es espacio de usuario, heredando todas las limitaciones que ello implica. Esto significa que los programas que quieran usar este protocolo de comunicación deben diseñarse específicamente para ello y emplear la interfaz que ofrece para llevar a cabo la comunicación en sí. Debido a esto, los programas comerciales que acceden a la red de manera estándar, como por ejemplo el Secure Shell (SSH), no pueden usar este tipo de red para comunicar con los distintos nodos que la constituyen.

La figura 1.2 (izquierda y centro) ilustra las diferencia entre los tipos de comunicación que se pueden llevar a cabo mediante el funcionamiento estándar de un sistema operativo normal y las que, hasta este momento, permite *RT-WMP*. La figura 1.2 (derecha) muestra, sin embargo, el sistema que se quiere obtener con el desarrollo de este proyecto.

La configuración actual impone limitaciones importantes en algunas situaciones, como por ejemplo, aquellas en las que se quiere acceder remotamente a uno de los nodos con el cual no se tenga posibilidad de conexión directa.

Los objetivos principales de este proyecto serán, por tanto, permitir la comunicación de forma transparente de cualquier tipo de programa que se ejecute en espacio de usuario y que

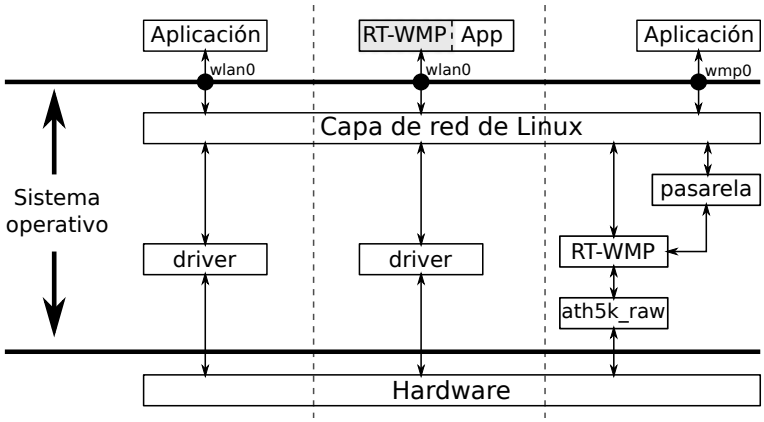


Figura 1.2: **Comunicaciones con y sin RT-WMP**

utilice el protocolo *IP* en la capa de red (en la práctica, casi la totalidad de las aplicaciones). Esto, además, abre la puerta a la posibilidad de poner en comunicación las redes *RT-WMP* con redes *IP* través de una pasarela y garantizar, por ejemplo, el acceso a Internet de nodos que se encuentren a muchos saltos de distancia del nodo que ejecute la pasarela, como ilustra la figura 1.3. La solución pasa por adaptar el protocolo al espacio de *kernel* y extender sus funcionalidades mediante una interfaz virtual adicional, que los programas puedan utilizar de forma transparente como cualquier otra.

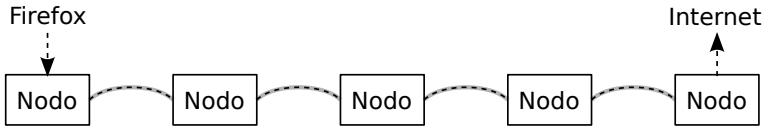


Figura 1.3: **Nodo accediendo a internet mediante otro nodo distante**

Además, se quiere mantener la posibilidad de utilizar el protocolo *RT-WMP* como medio para establecer comunicaciones con requisito de tiempo real de la misma forma en la que se permitían antes aprovechando, sin embargo, la mayor eficiencia que supone el hecho de ejecutar el núcleo del protocolo en espacio de *kernel* en vez de usuario.

1.2.1 Ejemplos de aplicaciones

Una vez alcanzado este objetivo, podrían emplearse programas convencionales de comunicación en situaciones en las que se utilice *RT-WMP* para ofrecer cobertura, simplificando de esta manera las tareas de las personas que necesiten hacer uso de esta cobertura, como puede ser un equipo de emergencias.

Otro tipo de situaciones donde este sistema podrá resultar útil es en el despliegue de grupos de robots, donde estos no siempre muestran el comportamiento esperado; en este tipo de situaciones sería muy interesante poder establecer una conexión con uno de los robots y estudiar la situación o, si hiciera falta, reiniciar un proceso problemático. Este problema se solventa empleando la solución presentada en este proyecto, dado que la conexión se establecerá por medio de la red RT-WMP.

1.2.2 Otros objetivos

Además de los objetivos anteriores, se desea disponer de la posibilidad de emplear los mecanismos de asignación de prioridades y de calidad de servicio de los que dispone *RT-WMP*

con el tráfico *IP*, de forma que se puedan establecer diferentes prioridades para diferentes tipos de tráfico, en función de las necesidades de cada escenario, y emplear el mecanismo de calidad de servicio para aquellos tipos de tráfico que así lo requieran. De esta forma se les podría asignar a las conexiones *VoIP* una alta prioridad y tratarlas como tráfico con requerimientos de calidad de servicio, mientras que el tráfico web podría tener una prioridad media y el *P2P* prioridad baja para no interferir con el resto.

1.3 Estructura de la memoria

La estructura que se ha dado a esta memoria es la siguiente:

- En el capítulo 2 se realiza una descripción del punto de partida del proyecto, dando una visión general de la situación a su comienzo y presentando el protocolo RT-WMP.
- En el capítulo 3 se describe la solución implementada, comenzando por una descripción general, para continuar con una detallada y una explicación de los pasos seguidos durante la elaboración de este proyecto.
- En el capítulo 4 se presentan los experimentos realizados y se realiza una evaluación de resultados.
- En el capítulo 5 se exponen las conclusiones del proyecto y los posibles trabajos futuros que permitirán continuarlo.

Además, se incluyen los siguientes apéndices:

- El apéndice A es el manual de uso; en él se explican todos los aspectos relativos a la compilación, la configuración y la puesta en funcionamiento de todo el sistema.
- En el apéndice B se exponen los resultados de una comparativa entre las llamadas *ioctl* y *procfs* que será descrita en el capítulo 3.
- En el apéndice C se exponen los resultados de las pruebas de evaluación de rendimiento.

Capítulo 2
Trabajo previo

Una vez realizada la introducción y puesta en contexto del proyecto, se procede a explicar brevemente el estado del sistema a su comienzo y a realizar una pequeña introducción al protocolo *RT-WMP*, con el objetivo de que pueda alcanzarse con ello una mayor comprensión del trabajo realizado durante este proyecto.

2.1 Visión general

El esquema general del sistema al comienzo del proyecto puede verse en la figura 2.1.

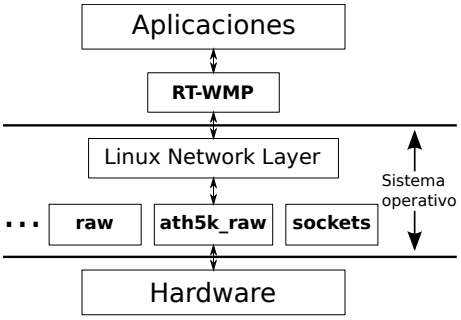


Figura 2.1: **Esquema inicial**

La capa de red de Linux (Linux Network Layer, LNL) es la capa del núcleo de Linux encargada de todos los aspectos relacionados con la gestión de las conexiones entre diferentes máquinas. En su interior se encuentra implementada, entre otras cosas, toda la pila de protocolos que permite la comunicación TCP/IP.

En este caso, cuando una aplicación envía datos por medio del protocolo RT-WMP, este procede a su envío por medio de su capa de comunicación de bajo nivel (ver más adelante), de la que existen múltiples implementaciones como puede verse en la figura 2.1. Por ejemplo, *sockets* emplea las operaciones normales de envío y recepción para la comunicación por medio del protocolo *UDP*, mientras que *ath5k_raw* configura la tarjeta inalámbrica y emplea *sockets* del tipo *SOCK_RAW* asociados a dicha tarjeta para realizar los envíos y recepciones.

En la recepción ocurre algo similar, los datos que proporciona el hardware pasan hasta la capa de red de Linux, que será la que se encargue de entregarlos a la aplicación correspondiente.

Atendiendo ya a los componente empleados durante este proyecto, se dispone de una versión modificada del controlador *ath5k* desarrollada durante la realización de un PFC anterior. Este

controlador permite el funcionamiento de varios modelos de tarjetas inalámbricas basadas en *chipset Atheros* de bajo coste y ampliamente difundidos. Esta versión se encuentra adaptada para los requisitos de comunicación de *RT-WMP* y recibe el nombre de *ath5k_raw* por el hecho de que transmite datos *ethernet* en crudo, de forma similar a si la comunicación se realizase por medio de un cable.

También se dispone de una implementación del protocolo *RT-WMP* diseñada para ser compilada para diferentes plataformas. Cabe señalar que de aquí en adelante, al hablar de plataformas refiriéndose a la implementación del protocolo no se habla de diferentes sistemas operativos ni de diferentes tipos de procesadores, sino de la forma que tomará el protocolo al ser compilado. En este sentido, al comienzo del proyecto se contaba con dos plataformas, *linux_us* y *MaRTE_OS*, que permiten al protocolo funcionar sobre GNU/Linux en el espacio de usuario, en forma de biblioteca, y sobre MaRTE OS [2] respectivamente.

Asimismo, *RT-WMP* cuenta con diferentes modos de realizar la comunicación de bajo nivel, es decir, el envío y la recepción de los datos. El principal de estos modos, por ser el más empleado, es el que hace uso del controlador *ath5k_raw*.

2.2 Protocolo RT-WMP

RT-WMP es un protocolo para *MANETs* diseñado para soportar tráfico de tiempo real estricto y que funciona sobre dispositivos que emplean el protocolo 802.11. Soporta cambios frecuentes en la topología de la red gracias al intercambio de una matriz calidad de los enlaces entre los nodos y prioridad estática de mensajes. Además, el protocolo está completamente descentralizado y emplea un esquema de paso de testigo para garantizar tiempos límite de transmisión.

Cabe destacar también el hecho de que el número de nodos que conforman una red *RT-WMP* es fijo y se establece al crearla, es decir, al configurar e iniciar el protocolo en cada uno de los nodos.

2.2.1 Visión general

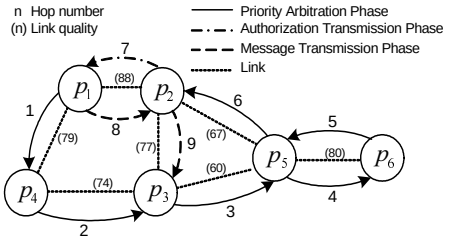
En una red *RT-WMP* todos los nodos se comunican empleando un único canal de radio compartido. En la versión básica del protocolo, cada nodo tiene una cola de transmisión y otra de recepción, ambas con prioridades. Cada mensaje intercambiado entre los nodos tiene un nivel de prioridad entre 0 y 127, siendo 127 la máxima prioridad. Los mensajes con la misma prioridad se almacenan en orden FIFO (First in, first out, primero en entrar, primero en salir).

Cuando una aplicación necesita trasmitir a otro nodo un mensaje, lo añade a la cola de transmisión. El proceso de *RT-WMP* extrae el mensaje de la cola y lo transmite al nodo destino a través de la red. El nodo destino encola este mensaje en la cola de recepción y es la aplicación destino la que finalmente los extraerá.

2.2.2 Fases del protocolo

El protocolo funciona en tres fases: fase de arbitraje de la prioridad (*Priority Arbitration Phase, PAP*), fase de autorización de la transmisión (*Authorization Transmission Phase, ATP*) y fase de transmisión del mensaje (*Message Transmision Phase, MTP*). Durante la PAP los nodos alcanzan un consenso sobre cual de ellos tiene el mensaje más prioritario (MPM) de la

red en ese momento. A continuación, en la ATP se envía una autorización para transmitir al nodo que tiene el mensaje con mayor prioridad. Por último, en la MTP este nodo envía el mensaje al nodo destino. En la figura 2.2 puede observarse un ejemplo con todas las fases.



Link Quality Matrix

	p_1	p_2	p_3	p_4	p_5	p_6
p_1	-	88	0	79	0	0
p_2	88	-	77	0	67	0
p_3	0	77	-	74	60	0
p_4	79	0	74	-	0	0
p_5	0	67	60	0	-	80
p_6	0	0	0	0	80	-

Figura 2.2: **Situación hipotética con su grafo y su LQM correspondiente**

Para alcanzar un consenso sobre cual de los nodos posee el mensaje con la mayor prioridad, en la PAP un token viaja por todos los nodos. El token contiene la información del nivel de prioridad del MPM de la red y de su propietario entre todos los nodos por los que ya ha pasado el token. El nodo que inicia la PAP envía un token indicando que él es el poseedor del mensaje con mayor prioridad, junto a dicha prioridad. Cada uno de los siguientes nodos compara la información del token con la de los mensajes en su propia cola. Si el nodo verifica que tiene un mensaje con mayor prioridad, modifica los datos del token antes de enviarlo al siguiente. El último nodo en recibir el token finaliza la PAP e inicia la ATP, puesto que ya conoce la identidad del aquel que tiene el MPM.

En la ATP, el nodo calcula un camino hasta el poseedor del MPM usando la información de la topología compartida entre los miembros de la red (ver el siguiente apartado) y envía un mensaje de autorización al primer nodo del camino. Este lo enviará al segundo y así hasta que la autorización alcance el nodo que tiene el MPM. En este punto comienza la MTP.

Esta fase se desarrolla de forma similar a la anterior. El nodo que ha recibido la autorización calcula el camino para alcanzar el destino y envía el mensaje al primer nodo de este camino. El mensaje sigue el camino y finalmente alcanzará su destino.

Las fases se repiten una tras otra; cuando termina la MTP el nodo destinatario del mensaje inicia una nueva PAP y el ciclo comienza. Cuando ninguno de los nodos tiene mensajes que transmitir las fases de autorización y de transmisión son omitidas y la fase de arbitraje de la prioridad se repite continuamente.

2.2.3 Matriz de calidad del enlace (LQM)

Para describir la topología de la red, *RT-WMP* define una extensión del grafo de conectividad de la red añadiendo valores no negativos a las aristas del grafo, es decir, pesos. Estos pesos son calculados en función de la señal de radio entre cada par de nodos y representan la calidad del enlace entre estos. Los pesos se representan en una matriz llamada matriz de calidad de enlace (*Link Quality Matrix, LQM*). Un ejemplo puede verse en la figura 2.2.

El elemento $[i, j]$ de la matriz representa la calidad del enlace entre el nodo i y el nodo j de la red y cada columna representa los enlaces de un nodo con sus vecinos. Aunque los enlaces pueden ser asimétricos (la señal de radio recibida por i cuando j transmite puede ser diferente de la recibida por j cuando i transmite), las diferencias son normalmente pequeñas y los enlaces se asumen simétricos. Los nodos emplean la matriz para seleccionar a que nodo pasar el token y para tomar decisiones sobre el mejor camino por el que enviar un mensaje.

Todos los nodos tienen una copia local de la LQM que se actualiza cada vez que se recibe una trama. Además, cada nodo es responsable de actualizar su columna en la matriz para informar al resto de nodos sobre los cambios locales de la topología.

2.2.4 Extensiones

El protocolo *RT-WMP* cuenta con cierto número de extensiones que pueden ser empleadas para añadir funcionalidades al comportamiento básico del protocolo. A continuación se procede a explicar brevemente aquellas extensiones que han sido empleadas en este proyecto:

- **multi_queue:** Esta extensión permite añadir el número de colas deseado al protocolo, de modo que en la recepción se decidirá a que cola corresponde cada mensaje del protocolo atendiendo al puerto que tenga asignado. Ésta extensión es especialmente útil cuando se requiere mantener varios canales de comunicación simultáneamente.

- **QoS:** Esta extensión permite transmitir mensajes atendiendo a los requerimientos de calidad de servicio por medio de la implementación sobre el protocolo RT-WMP de un mecanismo de prioridades dinámicas que tiene en cuenta su tiempo de expiración y el numero de saltos en la red hasta su destino [10, 9]. Para cumplir con su cometido añade una cola de transmisión y otra de recepción, ambas basadas en prioridades dinámicas.

- **broadcast:** Esta extensión permite emplear le dirección destino de un mensaje como un campo de bits, en el que cada bit designa a uno de los nodos de la red y el hecho de que este bit esté marcado o no indica si el mensaje debe ser enviado a dicho nodo o no. De esta forma, con una sola operación puede enviarse un mensaje a varios nodos de la red.

- **fake_lqm:** Esta extensión permite forzar una matriz de calidad de enlace con el objetivo de hacer pruebas de determinadas configuraciones como cadenas, estrellas, etc.

2.2.5 Arquitectura de la implementación

En la implementación del protocolo *RT-WMP* pueden distinguirse tres capas bien diferenciadas, del mismo modo que están representadas en la figura 2.3.

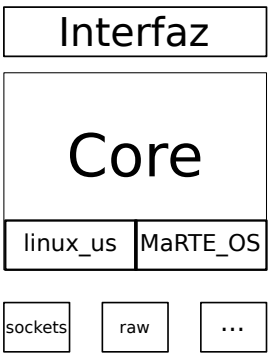


Figura 2.3: **Arquitectura del protocolo**

- La interfaz de *RT-WMP* con las aplicaciones. Esta capa es la que hace posible que las aplicaciones empleen el protocolo para comunicarse, encolando mensajes en la cola de transmisión y extrayéndolos de la de recepción. A su vez, ofrece funciones para controlar y monitorizar el comportamiento del protocolo.

- La capa central, donde se encuentra la implementación del protocolo en sí, tal y como ha sido descrito en los apartados anteriores. La mayor parte del código fuente es común, mientras que parte es específico para cada plataforma.
- La capa que se encarga de la comunicación de bajo nivel del protocolo. Esta capa contiene la implementación de una función de envío y otra de recepción de bajo nivel.

Gracias a este diseño, *RT-WMP* puede adaptarse con gran facilidad a diferentes tipos de comunicaciones de bajo nivel (principalmente diferentes tipos de hardware), simplemente sustituyendo la última de estas capas por otra que contenga la implementación adecuada de las funciones de transmisión y recepción.

Capítulo 3
Análisis y diseño del sistema

En este capítulo se realiza una descripción de la solución implementada, en primer lugar de forma general y después de forma más detallada, así como de los pasos seguidos a lo largo del proyecto.

3.1 Análisis

La división del sistema en módulos fue un paso directo, puesto que ya se disponía de una modificación del driver *ath5k* y del protocolo *RT-WMP* implementado como una librería. El único módulo completamente nuevo en el sistema es la propia pasarela.

El método de implementación de la pasarela, que fue un módulo del núcleo de Linux como se verá más adelante, vino impuesto por las funcionalidades que debía proveer, puesto que no existe ningún otro modo de satisfactorio de lograrlas.

En cuanto a la implementación interna de la pasarela, esta necesita gestionar varios flujos para los diferentes tipos de tráfico (*broadcast*, QoS, etc.). Por ello, en el envío los paquetes serán diferenciados según su tipo y se empleará la función apropiada de *RT-WMP* y en la recepción se tendrán hilos de ejecución esperando la llegada de nuevos datos en cada uno de estos flujos.

Las consideraciones de la fase de análisis sobre aspectos más concretos del sistema serán tratadas a lo largo de la descripción de la solución, en las secciones posteriores de este capítulo.

3.2 Introducción a la solución

En la figura 3.1 pueden observarse las diferencias entre una situación en la que un ordenador se comunica por medio de una red *IP* (izquierda) y como se comunica a través de una red *RT-WMP* usando el sistema construido a lo largo de este proyecto (derecha).

En la primera situación, la capa de red de Linux decide a través de que interfaz de red enviar los datos y esta, por medio del controlador de dispositivo, realiza el envío de los datos con la tarjeta de red. En la recepción, el driver proporciona los datos a la capa de red y esta los entrega a la aplicación que los espera o los descarta.

En nuestro caso en cambio, los datos *IP* generados por las aplicaciones atraviesan la pasarela, esta los envía al módulo *RT-WMP* y este a su vez los transmite por medio de *ath5k_raw*. Sin embargo, para las aplicaciones no hay diferencia, estas solo ven una interfaz de red por la que envían y reciben datos.

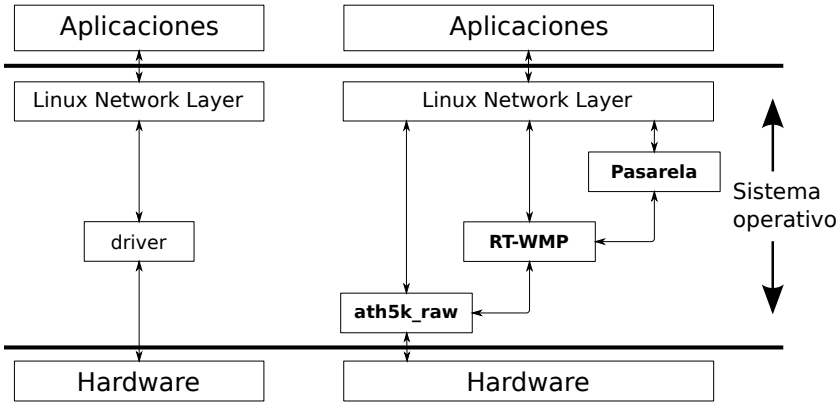


Figura 3.1: **Esquema general**

Además de permitir la comunicación *IP* por medio de redes *RT-WMP*, también se proporciona un mecanismo para la comunicación directa por medio de *RT-WMP*. Todos estos aspectos serán descritos con mayor detalle en secciones posteriores de este capítulo.

En lo que al sistema construido a lo largo de este proyecto se refiere, a simple vista puede observarse como ha sido añadido un componente, la pasarela, que se ocupa de tomar el tráfico *IP* generado localmente, encapsularlo dentro de mensajes *RT-WMP* y enviarlo al nodo correspondiente de la red, así como de recibir por medio del protocolo el tráfico *IP* encapsulado dentro de sus mensajes y enviándolo posteriormente a la capa de red de Linux para que esta lo haga llegar a la aplicación destino.

Además de añadir la pasarela, ha sido necesario realizar modificaciones en el controlador *ath5k_raw* y principalmente en *RT-WMP*, donde se ha añadido una tercera plataforma que funciona en espacio de *kernel* como un módulo. Estas modificaciones serán descritas en las siguientes secciones junto con una explicación pormenorizada de la pasarela.

3.2.1 Espacio de kernel

Antes de continuar con la explicación se va a realizar una breve explicación de la arquitectura del propio núcleo de Linux, para poder entender con mayor facilidad la siguientes secciones.

Linux divide la memoria del sistema en dos espacios completamente separados, conocidos como espacio de *kernel* (*kernel space*) y espacio de usuario (*user space*). El código que se ejecuta en el primero, el que forma parte del sistema operativo, tiene acceso íntegro y sin restricciones al sistema, mientras que el código ejecutado en espacio de usuario, las aplicaciones, tiene que utilizar los mecanismos que les proporciona el sistema operativo para acceder a los recursos del sistema.

Por otra parte, el núcleo de Linux es un núcleo monolítico híbrido, lo que significa que el código que tiene que ser ejecutado en espacio de *kernel* no necesita formar parte del núcleo. Esto significa que no hay que recompilarlo cada vez que se necesita añadir funcionalidades o añadir un nuevo controlador de dispositivo, sino que se proporciona un mecanismo por el cual estas funcionalidades pueden añadirse mientras este se encuentra en ejecución, en forma de módulos (Linux Loadable Kernel Module, LKM [7]).

3.3 Descripción detallada

3.3.1 Visión global

Para poder emplear programas diseñados para funcionar sobre *IP* en una red *RT-WMP* sin modificarlos es necesario que todo el sistema resulte transparente para estos. El modo empleado para lograr esto consistió en crear una interfaz de red virtual, que se comporta de la misma forma que cualquier otra interfaz *ethernet* desde el punto de vista de las aplicaciones, pero que internamente realiza las transmisiones por medio de *RT-WMP*. Como ya se ha comentado anteriormente, este es el papel de la pasarela.

La pasarela necesita realizar operaciones que solo pueden realizarse desde espacio de *kernel*, como la creación y configuración de una interfaz de red, por lo que hubo que implementarla como un módulo del *kernel* de Linux.

Lo mismo se hizo con *RT-WMP*, hecho que provocó gran parte de las numerosas modificaciones que se comentarán posteriormente en este capítulo.

3.3.2 Modificaciones al controlador

Para que *RT-WMP* pudiese obtener al ponerse en funcionamiento la interfaz de red controlada por *ath5k_raw* a utilizar, se realizaron las modificaciones oportunas en este último para que guardase la información de la primera interfaz de red registrada con dicho controlador y se implementó un procedimiento que permite obtener dicha información.

Por otro lado, se decidió realizar la comunicación entre *RT-WMP* y *ath5k_raw* directamente, sin pasar por la capa de red de Linux, evitando así el indeterminismo temporal que ello implicaría (algo indeseable al tratarse de un protocolo de tiempo real) y acelerando las comunicaciones al haber reducido las operaciones intermedias.

Las implicaciones que esta decisión tuvo en cuanto a la transmisión fueron mínimas, puesto que simplemente hubo que realizar el envío de los datos desde *RT-WMP* directamente, invocando la función de transmisión de la interfaz *ath5k_raw* en lugar de enviarlos a la capa de red de Linux por medio de la función correspondiente. Una representación de esta situación puede verse en la figura 3.2.

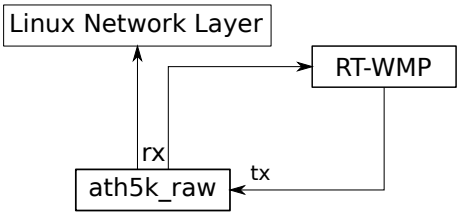


Figura 3.2: **Comunicación entre ath5k_raw y RT-WMP**

Antes de modificarlo, el controlador trataban los datos provenientes de la red de la forma apropiada y los enviaba a la capa de red por medio de la función definida en el núcleo de Linux a tal efecto. Una vez echo esto, el protocolo no tenía necesidad de almacenar estos datos a la espera de que la aplicación correspondiente los solicitase, la capa de red de Linux lo hacía en su lugar. Sin embargo, para hacer llegar sus datos directamente al protocolo *RT-WMP* es necesario almacenarlos dentro del propio controlador a la espera de que el protocolo solicite su recepción por medio de una función implementada con ese propósito

En primer lugar hubo que distinguir, dentro de *ath5k_raw*, entre los paquetes pertenecientes al protocolo y el resto de paquetes, para de este modo seguir mandando a la capa de red aquellos ajenos *RT-WMP*, como se puede ver en la figura 3.2. Esto se consiguió fácilmente realizando una comparación sobre el campo que contiene el protocolo de los paquetes dentro de la cabecera del protocolo *ethernet*.

En segundo lugar, se implementó un mecanismo para almacenar los paquetes a la espera de ser solicitados por el protocolo *RT-WMP*. Mediante este se garantiza el acceso en exclusión mutua a los datos, dado que pueden existir accesos concurrentes, y se permite bloquearse a la espera de la recepción de nuevos datos, algo requerido por las funciones empleadas por *RT-WMP* para solicitar estos datos.

Con ello ya se disponía de todas las herramientas necesarias para realizar la comunicación desde y hacia el protocolo *RT-WMP* sin pasar por la capa de red de Linux, directamente con *ath5k_raw*.

Por último, *RT-WMP* utilizaba llamadas *ioctl* (ver más adelante) para configurar la interfaz *ath5k_raw* de acuerdo a sus necesidades, pero la función que atiende estas llamadas está diseñada para recibir los datos desde el espacio de usuario. La solución a esta situación pasó por crear una nueva función adaptando la anterior, de manera que permitiese realizar estas mismas configuraciones pero recibiendo los datos desde otro módulo, es decir, desde espacio de *kernel*.

3.3.3 Adaptación de RT-WMP

Como ya se ha comentado, las modificaciones en *RT-WMP* vinieron principalmente ocasionadas por la necesidad de que este pudiese funcionar como un módulo del *kernel*. Este entorno, de hecho, supone ciertas limitaciones y obliga a tener tener en cuenta factores que no se dan programando para espacio de usuario:

- No se dispone de las bibliotecas estándar de C. Gran parte de las funciones de las bibliotecas estándares de C tienen su equivalente en el *kernel*, por lo que en estos casos los cambios se limitan al nombre, pero no siempre es así.
- Por defecto, la aritmética de coma flotante no está soportada.
- Las divisiones con enteros de 64 bits no están soportadas de forma nativa en máquinas de 32 bits.
- En el espacio de usuario, no liberar la memoria dinámica que se reserva es una mala práctica de programación, puesto que se producen fugas de memoria hasta que el programa finaliza su ejecución; es grave pero no es algo dramático por lo general, puesto que esta memoria es liberada una vez que el programa finaliza. Sin embargo, en espacio de *kernel* esto no sucede, puesto que la memoria asignada dinámicamente es memoria del *kernel*, no se guarda información de que módulo es el que la ha reservado, por lo que si el módulo no se encarga de liberarla, esa memoria permanecerá marcada como ocupada hasta que se reinicie el sistema, una situación potencialmente catastrófica. Por ello, hay que asegurarse de que toda la memoria dinámica reservada se libera posteriormente.

Para solventar esta situación y que el código siguiese siendo válido para las dos plataformas para las que estaba implementado *RT-WMP*, se añadió una tercera.

Integración de las plataformas

La forma de hacer que el protocolo compilase y funcionase en dos sistemas diferentes era teniendo gran parte del código común para ambas y disponer de un directorio para cada plataforma con el código específico para cada una de ellas, de forma que era en el momento de la compilación cuando se decidía utilizar los ficheros para la plataforma requerida, por medio de los mecanismos de compilación condicional que provee el conjunto de herramientas del proyecto GNU (*autotools*).

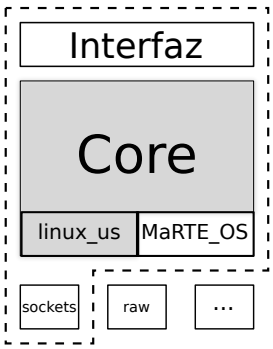


Figura 3.3: **Ejemplo del sistema compilado**

En la figura 3.3 puede observarse una situación en la que el protocolo se encuentra compilado para la plataforma *linux_us*. Esta forma de trabajar tiene la ventaja de que la mayor parte del código a las diferentes plataformas es común, no hay que mantener versiones diferentes del protocolo. Además, si se descubre un fallo en la implementación del *CORE* solo habrá que solucionarlo una vez, en lugar de tener que solucionarlo en cada una de las versiones.

Para las funciones que son diferentes entre las plataformas, cada una de ellas dispone de un fichero en el que se se encuentran definidos los mismos macros del preprocesador de C, pero en cada uno de ellos se definen como llamadas a las funciones correspondientes a la plataforma. De esta forma, en el código en común se emplean los macros y a la hora de la compilación el preprocesador se encarga de sustituirlos por las llamadas a función correspondientes.

Este mecanismo fue especialmente útil para solventar el primero de los problemas señalados. Se definieron para la nueva plataforma los macros que ya existían para las anteriores y se añadieron a las tres aquellos que fueron necesarios por la inclusión de la tercera, sustituyendo en el código en común las antiguas llamadas a función por estos nuevos macros.

```
#define MALLOC(size) malloc(size)                    /* En linux_us y MaRTE_OS */
#define MALLOC(size) kmalloc(size, GFP_KERNEL) /* En linux_ks */

/* Ejemplo de uso del macro */
d=(long *) MALLOC(n*sizeof(long));
```

Figura 3.4: **Diferentes definiciones de un macro y ejemplo de su uso**

Un ejemplo de lo anteriormente señalado es la reserva de memoria dinámica. Tanto en MaRTE OS como en Linux en el espacio de usuario se emplea la función *malloc*, pero esta función no está disponible en el espacio de *kernel*, sino que existe una similar llamada *kmalloc*. La solución a este problema fue añadir un macro con la función correspondiente en cada plataforma, como puede observarse en el ejemplo de la figura 3.4.

En las pocas ocasiones en las que no existían funciones similares que pudiesen ser empleadas, estas se implementaron completamente. Tal fue el caso de la función *fgets*, de la que se hablará más adelante, entre otras. Un ejemplo de implementación de la función *atoi* por medio de un macro puede verse en la figura 3.5.

```
#define atoi(s)                                          \
({                                                       \
   char *next;                                           \
   (int) simple_strtol(s,&next,10);                      \
})
```

Figura 3.5: **Implementación de una función inexistente en el kernel**

Coma flotante

Como se ha adelantado, las operaciones de coma flotante no están soportadas por defecto en espacio de *kernel*, esto es debido a una decisión de diseño del propio *kernel*. Se consideró que en este espacio la aritmética flotante era completamente innecesaria excepto en casos muy excepcionales, además de que guardar el contexto de la unidad de coma flotante (Floating-point unit, FPU) ocasionaría que los cambios de contexto fuesen considerablemente más lentos, por lo que se decidió no guardar el contexto de la FPU automáticamente. Sin embargo, se proporcionan dos funciones por medio de las cuales se puede realizar esta operación manualmente en caso de necesitarlo.

Estas funciones son *kernel_fpu_begin()* y *kernel_fpu_end()*, entre las que debe situarse el código que implique operaciones de coma flotante. Para introducirlas en el código se definieron los macros *FLOAT_OPS_START()* y *FLOAT_OPS_END()*, que serán sustituidos por el preprocesador para el caso de *linux_ks* por las funciones anteriores o simplemente eliminados para las otras dos plataformas.

De esta forma, se minimizó el uso de las operaciones de coma flotante en la medida de lo posible, sustituyéndolas por operaciones con números enteros en las situaciones en las que este cambio era posible o reordenando el código para minimizar el numero de operaciones en otros casos. Una vez hecho esto, simplemente se emplearon los macros descritos anteriormente para envolver todos los fragmentos de código con operaciones de este tipo restantes en el código fuente del protocolo.

Divisiones de 64 bits

Como ya se ha comentado anteriormente, las divisiones de enteros de de 64 bits no están soportadas en el *kernel* en las arquitecturas en las que no existe un soporte hardware para ello. Sin embargo, se dispone de un macro definido en el núcleo de Linux (*do_div*) que nos permite calcular simultáneamente la división y el resto resultante entre dos números.

A consecuencia de ello, hubo que localizar en el código del protocolo todas las operaciones de división que involucraban a los tipos *long long* y *unsigned long long* y sustituirlas por el macro *DO_DIV64*. Este macro se definió en el fichero señalado anteriormente, de forma que para las dos plataformas iniciales las divisiones se siguiesen realizando del mismo modo, pero que al compilar *RT-WMP* como módulo se emplease *do_div* para el cálculo.

Memoria dinámica

La responsabilidad de liberación de la memoria dinámica cuando se trabaja en espacio de *kernel* recae enteramente en el programador puesto que, como ya se ha comentado, ésta no se libera automáticamente en ningún momento. No liberar esta memoria supondría provocar fugas de memoria, dado que permanecería marcada como ocupada hasta un reinicio del sistema aunque realmente no lo está, una situación potencialmente catastrófica. Por lo tanto, se procedió a localizar en el código fuente todas las reservas de memoria y a asegurarse de que ésta fuese liberada en el momento oportuno.

Ficheros de configuración

El protocolo *RT-WMP* cuenta con dos ficheros de configuración, uno con opciones relativas a la configuración del propio protocolo y otro con opciones para la configuración de la tarjeta inalámbrica. Sin embargo, como ya se ha comentado anteriormente, al trabajar en el espacio de *kernel* no disponemos de las funciones convencionales de manipulación de ficheros. Aun así, el *kernel* dispone de sus propias funciones, pese a no ser tan sofisticadas como las proporcionadas por las bibliotecas de C.

Además, el código encargado del tratamiento de los ficheros de configuración no es común, sino que es específico para cada plataforma, hecho que facilitó la creación de las funciones correspondientes a la nueva plataforma.

Como las operaciones relacionadas con la lectura de datos eran considerablemente rudimentarias, se implementó con ellas un equivalente a la función *fgets* disponible en espacio de usuario. De esta forma se pudo reutilizar buena parte del código empleado en espacio de usuario para la lectura de los ficheros de configuración, contando con la ventaja de que este código llevaba tiempo siendo utilizado, por lo que la probabilidad de que contuviese un error era menor.

Los ficheros de configuración en el espacio de usuario se sitúan en un directorio dentro del *HOME* del usuario, pero al trabajar en el espacio de usuario se decidió situarlos en un lugar más apropiado para tal situación, como es el directorio */etc*. Por tanto, los ficheros de configuración para el protocolo cuando este se compila como un módulo se encuentran en */etc/rt-wmp/*.

Extensiones

Hasta ahora se ha hablado de las modificaciones realizadas a la implementación del protocolo pero, como ya se ha comentado anteriormente, el protocolo cuenta con una serie de extensiones. Las cuatro extensiones explicadas en el capítulo anterior (*multi_queue*, *QoS*, *broadcast* y *fake_lqm*) sufrieron el mismo tipo de modificaciones que el resto del código, con el objetivo de poder hacer uso de ellas cuando el protocolo fuese compilado como un módulo del *kernel*.

Además, el sistema de extensiones de *RT-WMP* estaba diseñado para que al cargar una extensión se reservase la memoria necesaria para sus estructuras y se definiesen las funciones que se debían invocar en determinados momentos durante la ejecución del protocolo. Sin embargo, no recibían un aviso cuando el protocolo finalizaba, por lo que no tenían forma de liberar su memoria. Para solucionarlo se añadió al sistema de extensiones la posibilidad de definir una función que se invoca en el momento de la detención de protocolo, mediante la que estas pueden liberar la memoria reservada durante su carga para evitar fugas de memoria o deshacer cualquier otra operación que necesite ser deshecha.

3.3.4 Pasarela

La pasarela es el módulo que se encarga de permitir las conexiones IP de forma transparente sobre redes construidas con el protocolo RT-WMP y, como ya se ha indicado anteriormente, esto lo consigue creando una interfaz de red virtual que lo enmascara.

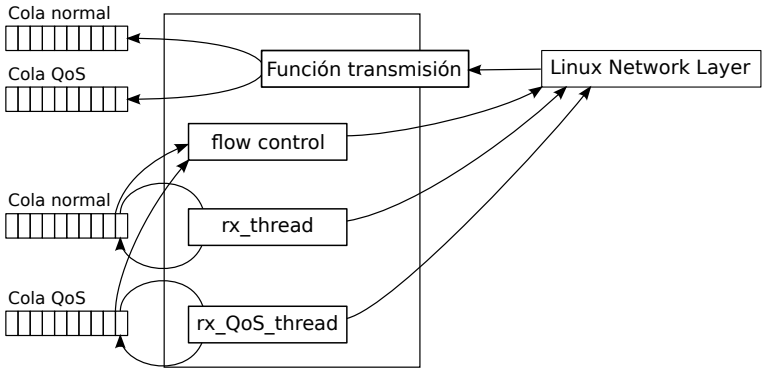


Figura 3.6: **Esquema general de la interfaz creada por la pasarela**

En la figura 3.6 pueden observarse los componentes principales de la interfaz que crea la pasarela: la función de transmisión, que distingue según el tipo de tráfico, y los diferentes hilos de ejecución. Las colas representadas en la figura son diferentes colas de transmisión y de recepción de RT-WMP.

Al cargar el módulo de la pasarela se crea y registra esta interfaz en el sistema y se realizan las configuraciones oportunas, como puede ser la longitud de su cola de transmisión o su unidad máxima de transferencia (Maximum Transfer Unit, MTU). Todos estos componentes serán descritos en los siguientes apartados.

Puesta en funcionamiento

A la hora de poner en funcionamiento la interfaz de la pasarela, se le asignará una dirección *IP* acorde a la subred de las máquinas conectadas a la red RT-WMP, de modo que pueda realizarse la comunicación *IP*. Para simplificar la puesta en funcionamiento del sistema, se decidió emplear esta dirección para configurar el número de nodos de la red y el identificador de nodo al lanzar el protocolo *RT-WMP*. Teniendo en cuenta que una dirección *IP* (versión 4) está compuesta por 4 bytes, se ignoran los dos bytes más significativos, el tercero se tomará como el número de nodos de la red y el cuarto se usará para el identificador del nodo (teniendo en cuenta que las direcciones *IP* para máquinas comienzan en 1 y los identificadores de *RT-WMP* comienzan en 0). La figura 3.7 muestra una representación gráfica de la correspondencia entre *IP* y *RT-WMP*.

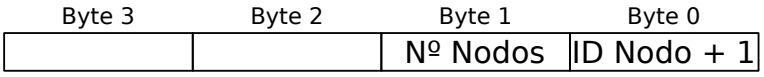


Figura 3.7: **Dirección IP / Datos de nodo RT-WMP**

De esta forma, si en una máquina se configura la interfaz virtual con la dirección *IP* 192.168.3.1 significará que la red *RT-WMP* está compuesta por tres nodos (incluida la propia máquina) y que la máquina tendrá el identificador 0 en esa red.

Por tanto, la función invocada al poner en funcionamiento la interfaz se encarga de:

- Generar aleatoriamente la dirección física de la interfaz.

- Obtener el número de nodos de la red RT-WMP y el identificador de nodo de la máquina en función de la dirección IP asignada a la interfaz e inicializar RT-WMP con ambos datos.
- Cargar las extensiones necesarias del protocolo.
- Leer el fichero de configuración con la información de las prioridades para los diferentes tipos de tráfico (ver más adelante).
- Poner en funcionamiento RT-WMP y todos los hilos de ejecución necesarios
- Inicializar la cola de transmisión para poder empezar a usar la interfaz.

Al detener la interfaz se invoca una función que se ocupa de ordenar detenerse a todos los hilos, detener la cola de transmisión y finalizar *RT-WMP*, es decir, se ocupa de deshacer las operaciones realizadas por la función de carga que necesitan ser deshechas.

Transmisión

La función encargada de la transmisión de datos es uno de los elementos más importantes de la pasarela, puesto que es la que se encarga de enviar los datos *IP* encapsulados dentro de *RT-WMP*, la función principal de la pasarela.

La primera tarea de esta función es asegurarse de que los datos a ser enviados pertenecen al protocolo *IP*, en otro caso se descartan. Una vez se sabe que el paquete a ser enviado es un paquete *IP*, se procede a obtener su destinatario dentro de la red *RT-WMP* por medio de su *IP* destino. En este paso pueden darse tres casos distintos en esta situación:

- La dirección pertenece a otra red, por lo que el paquete será enviado al nodo número 0, que hace de pasarela entre la red *RT-WMP* y el mundo exterior.
- La dirección *IP* pertenece a la red local pero el identificador de nodo obtenido a partir de dicha *IP* no pertenece a ningún nodo de la red (se sale de rango). En este caso se descarta el paquete.
- La dirección pertenece a la red y el identificador de nodo es válido, por lo que el paquete será enviado a dicho nodo.

En la pasarela se ha creado una correspondencia directa entre direcciones *IP* locales e identificadores *RT-WMP*, pero cuando los envíos son al exterior de la red ésta no sirve. La *IP* destino de tal envío será una externa a la red y la LNL utilizará el protocolo de resolución de direcciones [5] (Address Resolution Protocol, ARP) para obtener la dirección física de la máquina que hace de pasarela de la red con el exterior.

Una forma de solucionar este problema hubiese sido implementar una especie de *ARP* con el que se obtuviesen identificadores *RT-WMP* a partir de direcciones físicas, mediante el que se hubiese podido obtener el nodo destino del paquete mediante su dirección física de destino, pero esto hubiese hecho los envíos más lentos y hubiese añadido carga a la red, además de la complejidad de la propia solución.

Por esta razón, se tomó la decisión de fijar el nodo 0 como nodo que hace de pasarela de la red RT-WMP con el exterior y dejar para un posible trabajo futuro la posibilidad de añadir la opción de configurar este nodo manualmente o implementar algún mecanismo como el que acaba de ser descrito.

Después del nodo destino se establece el nodo origen (el propio nodo), la prioridad y el puerto por defecto, se copia el paquete dentro de los datos del mensaje y se procede a su envío, tras haber actualizado las estadísticas de transmisión de la interfaz. Justo antes de proceder al envío del mensaje existe un fragmento de código que se encarga del control de flujo y que será descrito más adelante.

Por último, la función encargada del envío del mensaje establece las propiedades específicas para cada tipo de mensaje, incluidos puerto y prioridad, y lo envía por medio de *RT-WMP*, también utilizando diferentes funciones según el tipo de mensaje, dependiendo de si debe emplearse la función de envío normal del protocolo o una función definida por una extensión, como sucede con los mensajes *QoS*.

Hilos de ejecución

Por otra parte, la interfaz cuenta con varios hilos de ejecución además del principal, como puede observarse en la figura 3.6. Uno de ellos pertenece al mecanismo de control de flujo y el resto se encargan de realizar las recepciones de los diferentes tipos de tráfico. Cada uno de ellos se encarga de la espera en uno de los canales de recepción. Una vez recibidos los datos por medio de *RT-WMP*, estos son formateados adecuadamente y enviados a la capa de red de Linux para que esta pueda a su vez hacerlos llegar a la aplicación correspondiente. En la figura 3.8 puede verse un ejemplo de este fichero.

```
# PROTOCOL DIRECTION PORT PRIORITY TRAFFIC
TCP TO 22 50 NORMAL
ICMP BOTH 0 20 NORMAL
UDP FROM 10019 10 QoS
```

Figura 3.8: **Ejemplo del fichero de configuración de la pasarela**

Fichero de configuración

Por último, la pasarela cuenta con un fichero de configuración que es leído a la hora de poner en funcionamiento la interfaz y que contiene el modo de envío y la prioridad de diferentes tipos de tráfico, identificados por su protocolo de la capa de transporte y su puerto. Puede obtenerse más información sobre este fichero de configuración consultando el apéndice A.

Los datos obtenidos de este fichero se emplean en la función de envío para decidir el modo de enviar los mensajes y la prioridad a asignarles. Aquellos mensajes para los que no se encuentra una configuración son enviados con la prioridad asignada por defecto y por medio del envío normal de *RT-WMP*.

3.4 Desarrollo del proyecto

En esta sección del capítulo se detalla la serie de pasos seguidos en el proceso de implementación de la solución, añadiendo un orden temporal a los elementos descritos en la sección anterior y proporcionando una visión más global para proceder a la explicación de las decisiones de implementación que involucraron a varios elementos del sistema al mismo tiempo. Pese a que los pasos aparecen descritos de forma secuencial por claridad, a la hora de la implementación algunos de ellos fueron desarrollados en paralelo.

3.4.1 Interfaz sobre ath5k_raw

El primer paso de la implementación consistió en construir directamente sobre el controlador *ath5k_raw* una interfaz virtual, que en el futuro pasaría a convertirse en la pasarela.

Los objetivos de este primer paso eran dos:

- Construir un esqueleto de interfaz virtual sobre el que poder trabajar posteriormente para desarrollar la pasarela.
- Implementar gran parte de las modificaciones necesarias dentro de *ath5k_raw*. En particular, las funciones para la transmisión y recepción de datos directamente desde otro modulo.

Al proceder de este modo se minimizó en cada paso el número de elementos involucrados, suavizando la toma de contacto con el sistema y, sobre todo, facilitando la búsqueda de errores en las primeras fases de la implementación, al contar únicamente con dos elementos en el sistema.

3.4.2 Adaptación de RT-WMP como módulo del kernel

El siguiente paso de la implementación consistió en portar el protocolo *RT-WMP* a espacio de *kernel*. Ya se ha explicado en la sección anterior que esto consistió en añadir una tercera plataforma al protocolo, llamada *linux_ks*, mediante la que este pudiese compilarse como un módulo del *kernel* de Linux y las implicaciones que esto conllevó, por lo que no se volverá a incidir en ello.

En lo que a la compilación se refiere, *RT-WMP* emplea *Autotools* para general los ficheros empleados en la compilación (*Makefiles*), mientras que la compilación de un módulo del núcleo de Linux se realiza por medio del mismo sistema por el que se compila el propio núcleo.

De esta forma, hubo que integrar el sistema de compilación para el módulo con el sistema existente para que al compilar el protocolo para la plataforma *linux_ks* se empleasen las herramientas adecuadas.

Con tal objetivo, cuando se compila habiendo seleccionado la plataforma *linux_ks* se invoca al sistema de compilación del núcleo pasando como parámetro el directorio en el que se encuentra el código fuente del módulo. Al hacer esto, este sistema busca en dicho directorio un fichero con el nombre *Kbuild*, que contendrá la información necesaria para construir el módulo [3, 1]. Con esta información el sistema compila todos los ficheros fuente necesarios y genera el módulo.

3.4.3 Integración de los tres módulos

Una vez se completaron los dos pasos anteriores, se procedió a la integración de los tres módulos para que funcionasen conjuntamente, es decir, se procedió a situar el módulo de *RT-WMP* entre la interfaz virtual y *ath5k_raw*.

Por un lado, se hizo que el módulo del protocolo *RT-WMP* comunicase directamente con *ath5k_raw* para las transmisiones y recepciones de datos y se implementó en este último el mecanismo que se ha descrito anteriormente para distinguir entre el tráfico *RT-WMP* y el resto y la función para que el protocolo pudiese configurar la tarjeta inalámbrica. Además, se hizo que al iniciar el protocolo, este pusiese en funcionamiento la interfaz *ath5k_raw* si no lo estaba ya y que restaurase su estado anterior al finalizar su ejecución.

Por otro, se procedió a enlazar la interfaz virtual con el módulo de *RT-WMP* por medio de las funciones que este último ofrece como interfaz, de forma que esta pudiese configurar el protocolo al comienzo como se ha descrito anteriormente y pudiese enviar datos a través de este.

Otro aspecto importante y que se ha ignorado hasta este momento es el que hace referencia a la visibilidad de las funciones de un módulo para que puedan ser usadas por otro. Para que un módulo pueda emplear una función definida en otro, esta debe ser exportada explícitamente por medio del macro *EXPORT_SYMBOL*. Por tanto, hubo que emplear dicho macro con todas las funciones tanto de *ath5k_raw* como de *RT-WMP* que necesitaban ser empleadas desde otro módulo.

Una vez completado este paso, el sistema ya podía emplearse para establecer comunicaciones *IP* sobre redes *RT-WMP* con aplicaciones convencionales.

3.4.4 Ampliación de las funcionalidades de la pasarela

Una vez se dispuso de una versión de la pasarela con las funcionalidades básicas para permitir establecer conexiones *IP*, se procedió a ampliar estas funcionalidades para permitir gestionar las prioridades en función del tipo de tráfico con ayuda del mecanismo de prioridades de *RT-WMP* y para poder enviar con requerimientos de calidad de servicio aquellos datos que así lo necesitasen por medio de la extensión de *QoS*.

Además, se implementó un mecanismo por medio de la extensión *broadcast* de *RT-WMP* para gestionar los envíos de este tipo de tráfico. De esta forma, cuando se envía un paquete en *broadcast* a la red local, la pasarela lo detecta por medio de la dirección *IP* y hace uso de dicha extensión para hacerlo llegar a todos los nodos de la red con un solo envío.

Además de dotar a la pasarela de los mecanismos necesarios para la gestión de los diferentes tipos de tráfico, también se implementó un mecanismo de control de flujo. Éste se implementó haciendo uso de una función que proporciona el *kernel* para informarle a la capa de red que debe parar el envío de paquetes de las aplicaciones a la interfaz, para permitirle vaciar sus colas.

De esta forma, cuando una de las colas de transmisión del protocolo cuanta con un número reducido de espacios libres se invoca esta función, de forma que las aplicaciones no pueden mandar más datos hasta que la cola vuelva a iniciarse. Con el objetivo de volver a iniciar la cola, se dispone de un hilo de ejecución que la inicia periódicamente en caso de que esté parada. De esta forma, se le concede tiempo al protocolo para que gestione los datos a la espera de ser enviados antes de introducir datos nuevos.

Por otra parte, y con el objetivo de que el tráfico *RT-WMP* y el tráfico de la pasarela empleasen canales de comunicación diferentes, se empleó la extensión *multi_queue* para crear dos canales de comunicación adicionales, uno para el tráfico normal y el "broadcast" y otro para el tráfico con requerimientos de calidad de servicio. Además, para que esta extensión funcionase conjuntamente con la de *QoS* hubo que realizar modificaciones en ambas. Una vez hecho esto, los mensajes del tráfico *RT-WMP* pudieron emplear las colas normales del protocolo y los del tráfico de la pasarela las creadas por medio de *multi_queue*.

Parámetros

El módulo de la pasarela admite como parámetro al ser cargado las extensiones de *RT-WMP* que se quieren usar (*broadcast*, *QoS* y/o *fake_lqm*) además de *multi_queue*, que es cargada siempre puesto que la emplea la propia pasarela. En caso de no cargar la extensión necesaria para cierto tipo de tráfico, la función de envío simplemente enviará dichos paquetes como tráfico normal, cosa que puede suceder al no cargar la extensión de *QoS*, por ejemplo.

3.4.5 Acceso directo a RT-WMP

Pese a que una vez alcanzado este punto la comunicación por medio de la pasarela cumplían los objetivos del proyecto, se había perdido la posibilidad de acceder al protocolo *RT-WMP* directamente desde una aplicación cuando este se compila como un módulo, puesto que una aplicación no puede invocar libremente funciones definidas en espacio de *kernel*.

Estudio de soluciones

Con el objetivo de solucionar este problema, se plantearon principalmente dos posibles soluciones:

- Crear por medio de *procfs* [4, 8] (process filesystem) una serie de ficheros en el directorio */proc* por medio de los que se realizaría la comunicación entre las aplicaciones de usuario y *RT-WMP*.
- Emplear llamadas *ioctl* [11] (input/output control) personalizadas mediantes las que transmitir datos entre espacio de usuario y espacio de *kernel*.

Procfs es un pseudo sistema de ficheros en los sistemas operativos tipo *UNIX* que se emplea para permitir el acceso a la información del *kernel* sobre los procesos. En Linux *procfs* incluye, entre otras cosas y además de la información sobre los procesos, información sobre el hardware, el propio *kernel* y los módulos y acceso a las opciones del núcleo que pueden configurarse dinámicamente, dado que estos ficheros permiten tanto leer como escribir.

Ioclt es una llamada al sistema para operaciones específicas sobre dispositivos que no pueden efectuarse por medio de llamadas al sistema convencionales. Esta llamada al sistema toma como parámetros un descriptor de fichero (un *socket*) abierto y asociado con el dispositivo, un número de operación y los datos. Además de las operaciones definidas por defecto para todos los dispositivos, cada dispositivo puede definir las suyas propias.

Ambas opciones permiten realizar la conexión con el protocolo sin pasar por la capa de red de Linux, para no introducir indeterminismo, ambas permiten realizar esta comunicación sin tener permisos de superusuario al ejecutar la aplicación y, resumiendo, permiten realizar todas las operaciones necesarias.

Dado que ambas opciones cuentan con características similares para las necesidades planteadas, se procedió a realizar un estudio temporal de ambas, con el objetivo de comprobarlas prestaciones de ambas soluciones. Esto se consiguió midiendo el tiempo en el momento en que la aplicación realiza la llamada y comparándolo con el tiempo en que el código de dicha llamada comienza a ejecutarse.

Los resultados, que pueden consultarse en el apéndice B, fueron bastante más significativos de lo esperado, puesto que mostraron que una llamada realizada por medio de *procfs* era en media

aproximadamente diez veces más lenta que la realizada con *ioctl*. Por tanto, se decidió emplear *ioctl* para implementar el acceso a *RT-WMP* para las aplicaciones.

Implementación y biblioteca de acceso

Una vez decidido el método de acceso a emplear y dado que las llamadas *ioctl* deben ser realizarlas por medio de una interfaz de red, hubo que decidir si se creaba una interfaz a tal efecto desde el módulo del protocolo o se empleaba la interfaz de la pasarela. Dado que el crear la interfaz desde RT-WMP no hubiese significado ninguna mejora y sí hubiese implicado tener una interfaz de red adicional simplemente para una comunicación que podía realizarse por medio de otra, se decidió implementar el acceso desde la pasarela.

Para facilitar el desarrollo de programas que funcionasen sobre *RT-WMP* y con el objetivo de que estos pudiesen ser empleados para trabajar sobre *RT-WMP* funcionando como un módulo del *kernel* o como una librería con el menor número de cambios posibles, se implementó una biblioteca que enmascarase las llamadas *ioctl*. La interfaz de esta biblioteca es muy similar a la interfaz que ofrece *RT-WMP* a las aplicaciones cuando es compilada como una biblioteca, por lo que el código fuente de los programas puede permanecer casi sin cambios, mientras que internamente las diferentes funciones de la biblioteca se encargan de realizar las llamadas *ioctl* correspondientes.

En la figura 3.9 pueden observarse las diferencias entre una situación y la otra. Mientras que en espacio de usuario las aplicaciones invocan directamente las funciones de la interfaz de *RT-WMP*, cuando este se compila como un módulo las aplicaciones invocan las funciones de la biblioteca, que a su vez invocarán las funciones de *RT-WMP* pasando por llamadas *ioctl* de la pasarela.

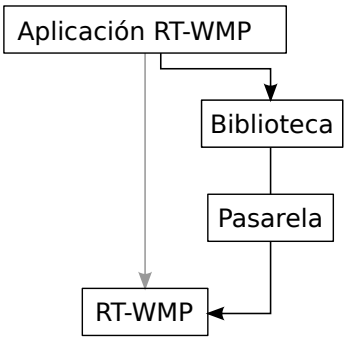


Figura 3.9: **Biblioteca de acceso a RT-WMP**

Por otro lado, hubo que tener en cuenta el hecho de que cada interfaz de red atiende las llamadas *ioctl* de forma secuencial, por lo que si una de ellas se bloquea, como es el caso de las recepciones bloqueantes, todas las demás llamadas tienen que esperar a que esta termine. Por tanto, no hubiese sido posible mantener varias comunicaciones simultaneas , ni siquiera una comunicación bidireccional.

Para conseguir evitar las llamadas *ioctl* bloqueantes, todas ellas recepciones, se decidió modificar el modo en que estas se llevaban a cabo; en vez de emplear un *socket* que se bloquease a la espera de la recepción de datos desde *RT-WMP*, se pasó a emplear dicho *socket* para bloquear el hilo de recepción por medio de llamadas *receive* propias de un *socket* del protocolo *UDP*, escuchando en un puerto concreto. Por su parte, la pasarela pasó a encargarse además de recibir estos datos desde *RT-WMP*, encapsularlos dentro de un paquete *UDP* adecuadamente

construido y enviarlos a la capa de red de Linux para que esta los hiciese llegar a la biblioteca, que estará esperando su recepción. Un esquema del encapsulado puede verse en la figura 3.10.

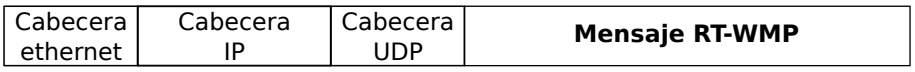


Figura 3.10: **Encapsulado de mensajes RT-WMP dentro de UDP**

La razón para emplear el protocolo UDP y no otro es su simplicidad, puesto que sus cabeceras contienen los campos estrictamente necesarios para hacer llegar los datos a su destino y no más, por lo que emplear este protocolo introduce menor sobrecarga que la que introduciría cualquier otro.

En lugar de encapsular los datos dentro de un paquete UDP podría haberse tomado la decisión de encapsularlos directamente dentro de un paquete *ethernet*, evitando así subir dos protocolos en la pila de protocolos TCP/IP. Esta decisión hubiese implicado utilizar en la biblioteca de acceso otro tipo de *socket*, "SOCK_RAW", que permite recibir desde una aplicación datos contenidos dentro de *ethernet*. El problema es que un programa necesita privilegios de superusuario para usar este tipo de *sockets*, por lo que todos los programas que empleasen *RT-WMP* deberían ser ejecutados como tal. De esta forma, se dejó como un posible trabajo futuro añadir esta opción de forma que se pueda decidir si usar un tipo de encapsulado u otro atendiendo a las ventajas y desventajas de cada uno.

3.4.6 Ficheros en /proc de RT-WMP

Pese a que se descartó el uso de *procfs* para el acceso a *RT-WMP* desde aplicaciones de espacio de usuario, se decidió emplearlo para crear una serie de ficheros en el directorio /proc mediante los que se pudiese consultar el estado en el que se encuentran los diferentes elementos del protocolo y modificar ciertas opciones en tiempo de ejecución, por medio de un simple interprete de comandos. Además, los retrasos que introduce *procfs*, razón por la que se descartó para el acceso a *RT-WMP*, no son significativos para la consulta y modificación de datos manualmente.

De esta forma, al poner en funcionamiento el protocolo estando este cargado como un módulo, se crean por medio de *procfs* una serie de ficheros dentro del directorio común */proc/rt-wmp* que permiten desde consultar el número de espacios libres u ocupados en las colas de transmisión y recepción del protocolo hasta modificar y comprobar varias opciones de configuración del protocolo. En la figura 3.11 puede observarse la lista de los ficheros creados por *RT-WMP*.

```
$ ls /proc/rt-wmp/
activeSearch            elementsInRXQueue   flowControl         instanceId
loopId                  mtu                 networkConnected    numOfNodes
rate                    timeout             cpuDelay            elementsInTXQueue
freePositionsInTXQueue  latestLQM           messageReschedule   netIT
nodeId                  primBasedRouting    serial              WCMult
```

Figura 3.11: **Listado de ficheros en /proc/rt-wmp**

3.4.7 Compilación conjunta de los 3 módulos

Una vez alcanzado este punto y teniendo en cuenta que la pasarela solo tiene sentido para la plataforma *linux_ks* de *RT-WMP*, se decidió integrarlos, de forma que siguiesen generándose

dos módulos separados pero la compilación se realizase conjuntamente. Para ello, se integraron ambas compilaciones y se añadió una opción al fichero de configuración de la compilación que permitiese decidir si se quiere compilar solo *RT-WMP* o también la pasarela.

Además, se tomó la decisión de permitir la ejecución del módulo de *RT-WMP* de forma autónoma, sin la pasarela encima, algo que hasta el momento no era posible puesto que era la propia pasarela quien configuraba y ponía en funcionamiento el módulo. Para ello, se añadieron una serie de parámetros al módulo de *RT-WMP* que permiten seleccionar su funcionamiento en modo autónomo y configurarlo del mismo modo que lo haría la pasarela. Este funcionamiento autónomo resulta interesante para nodos intermedios de la red, donde no hay interacción con el protocolo, sino que estos simplemente se encargan de transmitir el tráfico de unos nodos hacia otros; en tales casos la pasarela resulta innecesaria.

Una vez hecho esto, con el objetivo de facilitar el proceso de compilación y puesta en funcionamiento del sistema, se procedió a integrar también la compilación y la instalación de *ath5k_raw* con los otros dos módulos, para poder compilar e instalar los tres módulos conjuntamente. Para más detalles sobre como configurar y poner en funcionamiento el sistema, consúltese el apéndice A.

Capítulo 4
Evaluación y experimentos

Con el objetivo de evaluar el rendimiento de la implementación realizada a lo largo de este proyecto, así como de realizar una comparación entre la implementación del protocolo en espacio de *kernel* y la de espacio de usuario, se realizaron una serie de pruebas en el laboratorio y una prueba final en el interior del antiguo túnel ferroviario de Somport.

4.1 Entorno de pruebas

Todas las pruebas se realizaron empleando:

- El sistema desarrollado a lo largo de este proyecto funcionando en dos ordenadores personales. En una ocasiones la comunicación se realizaba entre ambos y en otras uno de ellos hacía de pasarela hacia internet.
- Una serie de nodos de comunicación desarrollados en un PFC anterior, ejecutando el protocolo *RT-WMP* sobre MaRTE OS conformando la estructura de la red por la que circulan los datos entre ambos ordenadores.

4.2 Pruebas en el laboratorio

En las pruebas en el laboratorio todos los elementos conectados a la red *RT-WMP* se encontraban muy cercanos, por lo que la red resultaba completamente conexa. Para obligar a que el tráfico entre ambos ordenadores circulase por medio de los nodos en esta situación se empleó la extensión *fake_lqm*, de forma que cada nodo interpretase que solo podía establecer comunicación con el nodo inmediatamente anterior a él y con el inmediatamente posterior, es decir, simulando una disposición lineal de acuerdo a su identificador. Por tanto, en todas las pruebas uno de los ordenadores tenía el identificador 0 y el otro el máximo identificador de la red.

Asimismo, durante las pruebas se ha empleado el programa *wmpSniffer* [12], que permite monitorizar comunicaciones *RT-WMP* para su posterior análisis.

Para consultar los resultados numéricos de las pruebas consúltese el apéndice C.

4.2.1 Comparativa entre linux_us y linux_ks

La primera de las pruebas consistió en la medición, con ayuda de *wmpSniffer*, del ancho de banda de la red para redes de diferente número de nodos, desde 2 (los dos ordenadores) hasta

7 (los ordenadores junto a los 5 nodos con MaRTE OS). Tanto en los ordenadores como en los nodos se lanzó un programa que generaba tráfico *RT-WMP* con destino y prioridad aleatorios, de forma que la red se encontrase saturada en el momento de realizar las mediciones.

El objetivo de esta prueba era comparar las prestaciones de ambas implementaciones. En la figura 4.1 puede verse una gráfica con los resultados de la prueba, donde se muestra para cada plataforma el ancho de banda obtenido para los diferentes tamaños de la red.

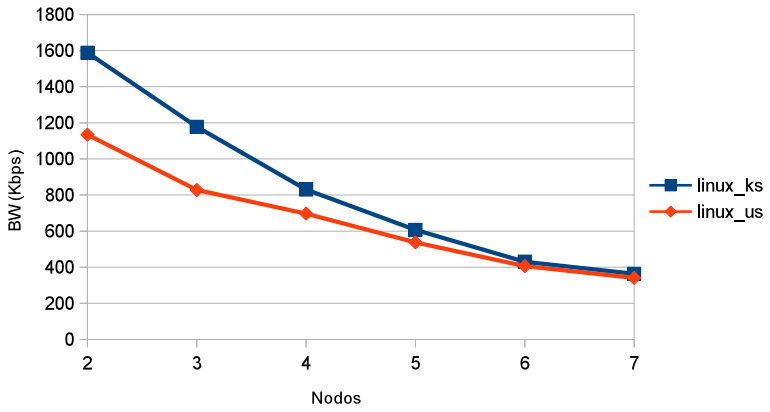


Figura 4.1: **Ancho de banda en redes de diferente tamaño**

Como puede observarse, se ha conseguido una ganancia notable en el ancho de banda, aunque esta disminuye conforme crece el tamaño de la red, alcanzándose una ganancia de un 5 % aproximadamente con el máximo tamaño. El hecho de las líneas converjan es lógico, puesto que conforme el tamaño de la red crecía los nodos que se añadidos funcionaban con MaRTE OS, por lo que ambos escenarios se iban pareciendo más.

Duración de una iteración en el caso peor

Con los resultados de esta prueba pudo analizarse también la duración de una iteración, es decir, el tiempo transcurrido desde que comienza una fase de arbitraje de prioridad hasta que comienza la siguiente, en el caso peor.

Este tiempo ya ha sido estudiado anteriormente y puede representarse con la siguiente fórmula: $(2N-3)t_t+(N-1)t_a+(N-1)t_m$, siendo N el número de nodos de la red, t_t el tiempo de transmisión de un token entre dos nodos, t_a el tiempo de transmisión de una autorización y t_m el tiempo de transmisión de un mensaje de máximo tamaño.

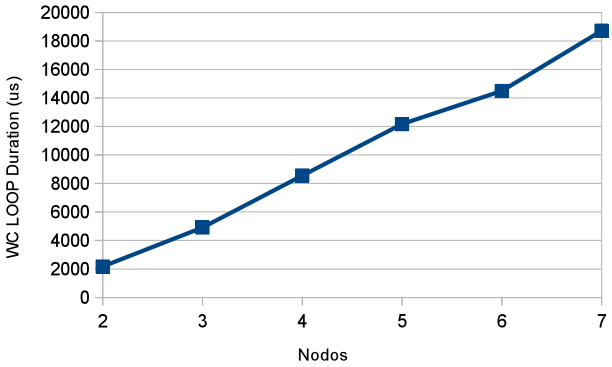


Figura 4.2: **Duración de una iteración en el caso peor**

En la figura 4.2 pueden observarse los resultados obtenidos para la plataforma *linux_ks* con diferentes tamaños de la red. Como puede verse, los tiempos crecen linealmente con el tamaño de la red, lo que concuerda con la formula teórica.

4.2.2 Equidad (Fairness)

En una segunda prueba, se configuraron todos los nodos de la red para emitir tráfico con la misma prioridad y poder observar así los retrasos en la entrega de los mensajes por medio de *wmpSniffer*.

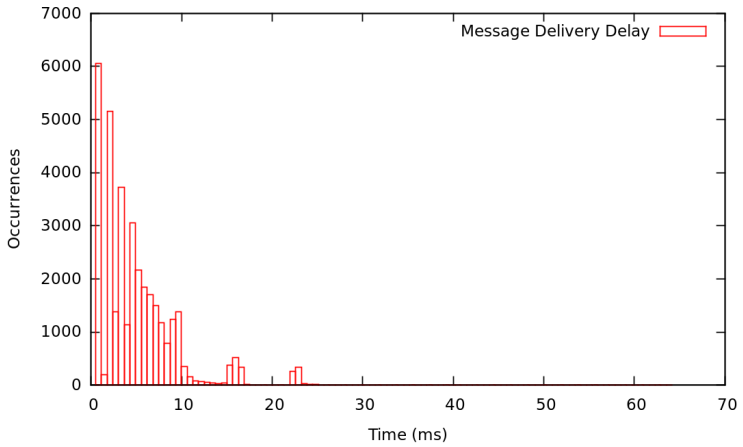


Figura 4.3: **Retraso en la entrega de los mensajes**

El la figura 4.3 se representan los mensajes agrupados por su retraso en la entrega, es decir, el tiempo que transcurren circulando por la red desde el origen hasta el destino. Como puede verse, los mensajes tienden a tener todos el mismo retraso en la entrega, lo que es de esperar puesto que todos ellos tienen la misma prioridad.

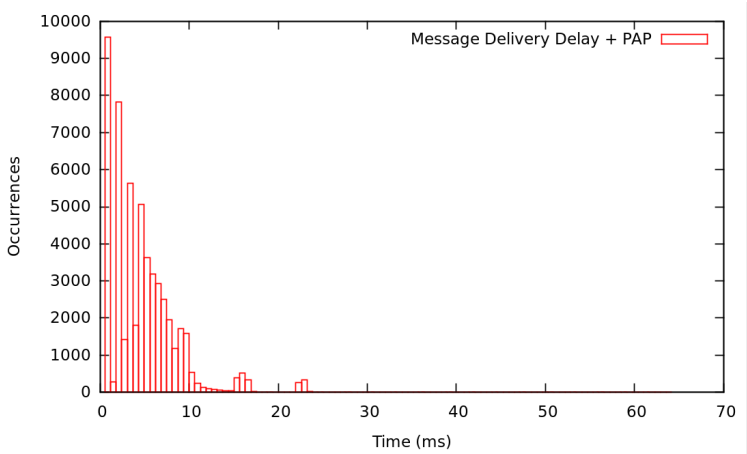


Figura 4.4: **Retraso en la entrega de los mensajes + Arbitraje**

En la figura 4.4 es similar a la anterior, pero se ha añadido el tiempo debido a la fase de arbitraje de prioridad, de forma que en esta gráfica los mensajes se agrupan según el tiempo desde que comienza el ciclo en el que van a ser enviados hasta que han alcanzado su destino. Al igual que en el caso de anterior, puede observarse como estos tiempos tienden a ser similares, debido una vez más a que todos los mensajes tienen la misma prioridad.

4.2.3 Pruebas con Skype

En esta prueba se configuró uno de los ordenadores como pasarela a Internet, conectándolo por cable a la red cableada, y con el otro ordenador se realizaron llamadas de voz mediante *Skype*, incrementando sucesivamente el tamaño de la red.

Pese a que el ancho de banda fue disminuyendo conforme aumentaba el tamaño de la red, como era de esperar, las comunicaciones pudieron mantenerse sin ningún tipo de problema.

4.2.4 Prueba completa

La última prueba realizada en el laboratorio se efectuó con el objetivo de comprobar el correcto funcionamiento de todos los elementos con la configuración que tendrían en la prueba de campo. Por tanto, la máquina que hizo de pasarela a Internet se conectó por medio de un teléfono móvil con 3G y se configuraron 4 nodos con MaRTE OS a la frecuencia adecuada.

4.3 Prueba de campo

Como ya se ha comentado anteriormente, la prueba de campo se efectuó en el túnel de Somport, cuya sección puede verse en la figura 4.5, un antiguo túnel ferroviario situado en la frontera entre España y Francia.

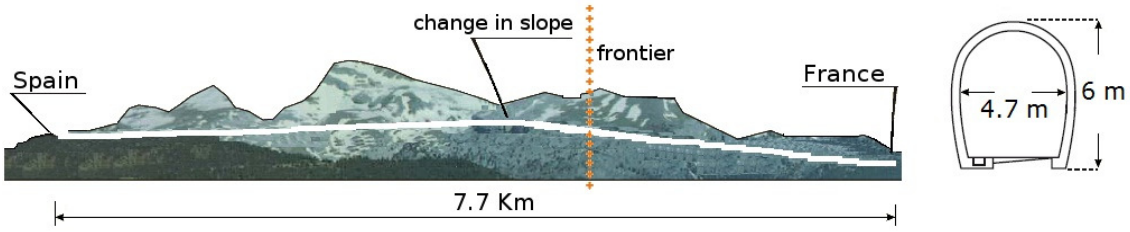


Figura 4.5: **Sección del túnel de Somport**

4.3.1 Preparación

En primer lugar, se situó un un ordenador portátil conectado a internet por medio de un teléfono móvil con 3G en la boca del túnel (figura 4.6) y se configuró para que fuese el nodo 0 de la red *RT-WMP*, es decir, la pasarela al exterior. El objetivo de esta prueba era demostrar la posibilidad de ofrecer cobertura de voz en el túnel por medio del sistema implementado durante el proyecto, usando aplicaciones convencionales para la comunicación.

A continuación, se situó el primero de los nodos con MaRTE OS (figura 4.6) a un kilómetro de la boca y se situaron los otros 3 nodos de forma que entre cada par consecutivo de nodos hubiese buena señal. De esta forma las distancias entre los nodos fueron de algo más de un kilómetro.

Una vez configurados los elementos estáticos de la red, se configuró un segundo ordenador portátil para que se conectase a la red *RT-WMP*, con lo que la red de 6 nodos estaba completa.

4.3.2 Pruebas

Tras asegurarse de que la conexión a internet en el segundo portátil por medio de la red *RT-WMP* funcionaba correctamente se procedió a efectuar las pruebas.

Figura 4.6: **Portátil que hace de pasarela y nodo intermedio**

Primera prueba

En la primera de las pruebas se mantuvo una conversación por medio del chat de voz de *Gmail* mientras se circulaba en con un coche desde la boca del túnel hasta varios kilómetros en su interior, de forma que el portátil tuviese que cambiar sucesivamente de nodo con el que mantenía la comunicación para estar conectado a la red.

Pese a que la comunicación de voz no fue perfecta, en especial cuando se estaba situado en una zona entre dos nodos, la conexión no se vio interrumpida y pudo mantenerse una comunicación bastante fluida pese a estar viajando por el túnel en el interior de un coche y a estarse realizando la comunicación por medio de una red 3G.

Segunda prueba

En la segunda de las pruebas se mantuvieron dos conversaciones con dos personas diferentes (una en Zaragoza y otra en Alemania); en esta ocasión en el exterior del vehículo. En esta prueba la comunicación se pudo realizar sin ningún problema y ambas partes de la comunicación escuchaban a la otra claramente y sin interrupciones de ningún tipo.

Última prueba

En la última de las pruebas se empleó el programa de *VoIP Skype* para realizar la comunicación, por realizar una prueba con un programa diferente. Los resultados de esta prueba fueron similares a los de la anterior, la comunicación se mantuvo con fluidez

Capítulo 5
Conclusiones y trabajo futuro

5.1 Conclusiones

Los objetivos de este proyecto eran, por un lado, permitir el uso de aplicaciones convencionales sobre redes *RT-WMP* y su interconexión con redes *IP* y, por otro, permitir que estas aplicaciones hagan uso de características de *RT-WMP* tales como la asignación de prioridades estáticas o la gestión de la calidad de servicio en las comunicaciones. Durante la realización de este proyecto se han ido cumpliendo satisfactoriamente todos estos objetivos.

Por un lado, se ha completado la implementación de la pasarela entre ambos protocolos, permitiendo de esta forma establecer comunicaciones *IP* sobre redes *RT-WMP*. El disponer de esta herramienta facilitará la realización de ciertas aplicaciones del grupo, puesto que se podrá acceder remotamente a equipos que estén comunicándose mediante *RT-WMP* sin interferir en su comunicación, como pueden ser grupos de robots desplegados en algún escenario.

Además, si la red se conecta a internet por medio de uno de los nodos, como se ha hecho en varias de las pruebas realizadas durante este proyecto, se podrá interactuar con ellos o verificar su correcto funcionamiento sin necesidad de desplazarse hasta su posición, tanto desde el puesto de mando de la propia aplicación como desde cualquier otra posición, a través de internet.

Por otro, se ha dotado a la pasarela de la posibilidad de emplear los mecanismos de prioridades estáticas y de calidad de servicio de los que dispone el protocolo, cumpliendo el último de los objetivos del proyecto. Mediante este mecanismo se pueden priorizar los tráficos importantes y evitar que otros menos importantes saturen la red.

Además de la consecución de los objetivos del proyecto, se han implementado algunas funcionalidades adicionales que se consideraron útiles para el sistema, como puede ser la creación de ficheros en el directorio /proc mediante los que consultar el estado del sistema y poder modificar ciertas opciones del protocolo. Asimismo, se implementó el mecanismo de acceso a *RT-WMP* mediante *ioctl* y la biblioteca para las aplicaciones, de forma que la consecuciones de los objetivos del proyecto no supusiese la pérdida de funcionalidades del sistema.

Por último, se validó el correcto funcionamiento del sistema desarrollado mediante de pruebas en un escenario real, verificando que todos los elementos del sistema funcionaban correctamente.

En un ámbito más personal, mi valoración del proyecto es muy positiva puesto que, además de alcanzarse con éxito los objetivos del proyecto, me ha permitido adquirir una valiosa experiencia trabajando con módulos del núcleo de Linux y con manejadores de dispositivo, así como con

ciertos aspectos de las redes de comunicación. Además, me ha permitido colaborar en un proyecto de software libre como es el protocolo *RT-WMP*.

5.2 Trabajo futuro

Aunque los objetivos del proyecto se han cumplido con éxito, existen funcionalidades que pueden ser añadidas al sistema y aspectos en los que se puede trabajar.

Un posible trabajo futuro del que ya se ha hablado durante la memoria es la posibilidad de modificar la implementación del acceso a *RT-WMP* por medio de *icmp* para las aplicaciones, realizándolo por medio de otro tipo de *sockets* que evitarían el encapsulado de las recepciones en paquetes *UDP*, aunque requerirían ejecutar los programas con privilegios de administrador.

Otro aspecto en el que se puede trabajar es en la configuración del nodo que hace de pasarela al exterior. Actualmente dicho nodo tiene que ser configurado con el identificador número 0 de *RT-WMP*, pero podría estudiarse el hacer dicha configuración más flexible.

En lo referente a las extensiones, existen varios aspectos en los que se puede trabajar. Por un lado, la extensión *multi_queue* es empleada en la actualidad por la pasarela para crear colas adicionales para su comunicación y no se le da posibilidad al usuario de crear las suyas propias. Por lo tanto, el trabajo consistiría en dar dicha posibilidad al usuario. Por otro, durante la realización de este proyecto se han adaptado las extensiones necesarias para ser empleadas con la plataforma *linux_ks*, pero *RT-WMP* cuenta con otras extensiones que también pueden ser adaptadas.

Un posible trabajo a más largo plazo consistiría en añadir más versiones de la capa de bajo nivel del protocolo, con el objetivo de soportar más tipos de tarjetas, como pueden ser las *Atheros* más modernas, que emplean el driver *ath9k* en lugar del *ath5k*.

Por último, alejándose ya de temas directamente relacionados con la implementación, existe la posibilidad de realizar una publicación con el trabajo realizado a lo largo de este proyecto.

Apéndice A
Manual de uso

En este apéndice se tratan los aspectos relacionados con la compilación, configuración y puesta en funcionamiento de los tres módulos que componen el sistema.

El sistema operativo necesario para compilar y ejecutar los tres módulos es GNU/Linux, con un *kernel* de Linux razonablemente moderno. Además, para la compilación es necesario tener instalados en el sistema las cabeceras del *kernel* (paquete *linux-headers* o similar) y los programas *autoconf*, *automake*, *make* y *gcc*.

A.1 Compilación e instalación

Pese a que el sistema completo está compuesto por tres módulos, el método de compilación e instalación ha sido diseñado para que todo se haga conjuntamente, evitando así complicaciones al usuario. Los comandos que se describen a continuación presuponen que se está situado en el directorio principal del protocolo y que se dispone de las herramientas necesarias para realizar las siguientes operaciones.

A.1.1 Previos

En primer lugar, se actualizan los ficheros de configuración necesarios por medio del comando *autoreconf* y se generan todos los ficheros *Makefile.in* a partir de los correspondientes *Makefile.am*, por medio del comando *automake -a*, instalando en el proceso los ficheros estándar que falten:

```
$ autoreconf
$ automake -a
```

Una vez hecho esto, está todo listo para proceder a la configuración de la compilación.

A.1.2 Configuración

En este paso se indica que debe compilarse *RT-WMP* para la plataforma *linux_ks*, dado que no es la plataforma por defecto.

```
$ ./configure --with-platform=linux_ks
```

Además, en caso de querer compilar solo *ath5k_raw* y *RT-WMP* y no la pasarela, se debe proporcionar el parámetro *–disable-ip-interface*.

Mediante la opción *–with-llcom* puede seleccionarse otra opción para la capa de bajo nivel, pero por el momento *ath5k_raw* es la única disponible para la plataforma *linux_ks*, por lo que es la que se toma por defecto, haciendo innecesario emplear este parámetro en esta plataforma mientras no se añadan más opciones.

Al final de este proceso se generan los ficheros *Makefile* necesarios para realizar la compilación.

A.1.3 Compilación

Una vez se han realizado todos los pasos anteriores, se puede proceder con la compilación:

```
$ make
```

Al finalizar esta operación se dispondrá de los módulos compilados, así como de la biblioteca de acceso a *RT-WMP* en caso de no haber deshabilitado la compilación de la pasarela.

A.1.4 Instalación

El único paso restante es la instalación en el sistema tanto de las módulos como de la biblioteca. Esta operación debe realizarse con privilegios de superusuario.

```
# make install
```

Una vez hecho esto el sistema se encontrará instalado en la máquina y listo para ser usado.

A.2 Configuración

Como ya se ha comentado a lo largo de la memoria, tanto el protocolo *RT-WMP* como la pasarela cuentan con ficheros de configuración mediante los que se pueden modificar los valores por defecto de determinadas opciones. Todos estos ficheros se encuentran en del directorio común */etc/rt-wmp*.

Todos los ficheros de configuración son simples ficheros que pueden editarse manualmente con cualquier editor de textos y que contienen una opción por linea. Además, aquellas lineas del fichero que no comiencen por una letra mayúscula son automáticamente descartadas, por lo que puede usarse esta característica para introducir los comentarios que se consideren oportunos.

A.2.1 RT-WMP

El fichero de configuración principal del protocolo se llama *rt-wmp.cfg* y puede contener opciones como el identificador del nodo, el número de nodos de la red o diferentes variables sobre el comportamiento del protocolo. El contenido de este fichero es el mismo que el empleado para la plataforma *linux_us*. En él puede encontrarse en cada linea el nombre de la opción junto con su valor separados por un espacio, como puede observarse en el siguiente ejemplo:

```
WMP_ADDRESS 0
N_NODES 5
```

Mediante esta configuración se indica a *RT-WMP* que es el nodo 0 de una red formada por 5 nodos. Aquellas opciones que no se configuran por medio del fichero tomarán su valor por defecto. Para más información consúltese el manual de RT-WMP.

A.2.2 Ath5k_raw ll_com

La capa de comunicación de bajo nivel del protocolo cuenta también con su propio fichero de configuración, llamado *linux_ks-ath5k.ll*. Este fichero es leído para efectuar la configuración de la tarjeta inalámbrica controlada por medio de *ath5k_raw* y en él puede indicarse la frecuencia de radio, el ratio y la potencia de la transmisión en la antena y su modo de funcionamiento. A continuación puede observarse un ejemplo del contenido de este fichero:

```
FREQ 5200
TXPOWER_DBR 15
RATE 60
ANTENNA_MODE AR5K_ANTMODE_FIXED_B
```

Con esta configuración se establece la frecuencia a 5200 Mhz, la potencia a 15 dBm, el ratio de transmisión a 60 * 100 kps y el modo indicado de funcionamiento de la antena.

A.2.3 Tráfico de la pasarela

La pasarela cuenta también con su propio fichero de configuración, llamado *interface.cfg*, en el que se encuentra las configuraciones relativas a los tipos de tráfico.

Cada linea del fichero debe contener cinco elementos separados por espacios:

- Protocolo de la capa de transporte. Puede ser *TCP*, *UDP* o *ICMP*.
- Sentido del tráfico con respecto al puerto. Puede ser *FROM* para el tráfico que tiene como origen el puerto indicado, *TO* para el tráfico que lo tiene como destino o *BOTH* para ambas.
- Puerto. Dado que el protocolo *ICMP* no tiene puertos, este campo se ignorará para este protocolo.
- Prioridad a asignar a los paquetes identificados con los tres campos anteriores. Puede tomar valores entre 0 y 127.
- Tipo de tráfico para los paquetes. Puede ser *NORMAL* o *QoS*.

A continuación puede observarse un ejemplo del contenido de este fichero:

```
# PROTOCOL DIRECTION PORT PRIORITY TRAFFIC
TCP TO 22 50 NORMAL
ICMP BOTH 0 20 NORMAL
UDP FROM 10019 10 QoS
```

Con esta configuración los paquetes *TCP* con destino el puerto 22 (conexiones a servidores *SSH*) tendrán prioridad 50, todo el tráfico *ICMP* tendrá prioridad 20 (el valor del puerto se ignora para ICMP) y los paquetes *UDP* con origen el puerto 10019 local tomarán la prioridad 10 y serán tratados como paquetes con requerimientos de calidad de servicio. El resto del tráfico será tratado como tráfico normal y tomará la prioridad por defecto. Además, en este ejemplo puede observarse que la primera linea es un comentario, empleado para indicar la composición de cada linea del fichero.

A.3 Puesta en funcionamiento

Como ya se ha indicado a lo largo de la memoria, el módulo de *RT-WMP* puede funcionar junto con la pasarela, de forma que puede ser usado para establecer comunicaciones tanto del protocolo como *IP*, o funcionar de manera autónoma, útil para los nodos internos de la red donde no se envían o reciben paquetes de ningún tipo, solo se retransmiten los de otros nodos.

A.3.1 Funcionamiento autónomo

Para ejecutar el módulo de *RT-WMP* en modo autónomo, debe proporcionarse el parámetro *autonomous=yes* al módulo en el momento de su carga, pues este se ejecuta por defecto esperando ser puesto en funcionamiento por la pasarela. El módulo admite otra serie de parámetros que solo son tenidos en cuenta cuando ese se ejecuta en este modo:

Por un lado, se le puede indicar al módulo por medio de los parámetros *node_id* y *num_nodes* el identificador de nodo y el número de nodos de la red respectivamente. En caso de no proporcionarse uno de estos parámetros se tomará el valor leído del fichero de configuración del protocolo.

Por otro, existe un parámetro para indicar por medio de una lista separada por comas las extensiones que queremos cargar al módulo. Estas extensiones pueden ser: *qos*, *broadcast*, *multi_queue* o *fake_lqm*. En caso de cargar la extensión *multi_queue* debe proporcionarse un parámetro adicional (*multi_queues*) indicando el número de colas que debe crear esta extensión.

De esta forma, al cargar el módulo del protocolo, este comprueba si se debe ejecutar en modo autónomo y de ser así, comprueba el resto de sus parámetros y se pone en funcionamiento de acuerdo a estos.

A.3.2 Con interfaz

Para poner en funcionamiento el protocolo con la interfaz debemos cargar el módulo correspondiente (*modprobe* se ocupará de cargar *ath5k_raw* y *rt_wmp*). Este módulo es más sencillo en lo que a sus parámetros se refiere, pues solo cuenta con uno, la lista de extensiones de *RT-WMP* a emplear, en la que pueden aparecer *qos*, *broadcast* y *fake_lqm* (*multi_queue* se carga siempre por las razones ya descritas en la memoria).

Una vez cargado este módulo, se dispondrá de una nueva interfaz de red llamada *wmp0*, se configura y emplea de igual modo que cualquier otra interfaz de red del sistema. Al ponerla en funcionamiento se obtendrán los datos del identificador de nodo y del número de nodos de la red a partir de la dirección *IP* asignada a la interfaz. Con estos datos se configurará y pondrá en funcionamiento *RT-WMP*.

A partir de este punto, podrá emplearse dicha interfaz para realizar comunicaciones *IP* o bien para comunicaciones *RT-WMP* por medio de la biblioteca proporcionada a tal efecto.

Apéndice B
Comparativa entre Ioctl y Procfs

Con el objetivo de decidir cual era le mejor opción para implementar el acceso al módulo *RT-WMP* desde las aplicaciones, se realizó un estudio temporal en el que se midió el tiempo transcurrido desde la realización de la llamada en la aplicación hasta el comienzo de la ejecución del código correspondiente en el módulo. Para ello se realizaron tres tandas de pruebas, una para *Ioctl* y dos para *Procfs*, una con operaciones de lectura sobre ficheros en /proc y otra con operaciones de escritura.

B.1 Mediciones

En la tabla B.1 se exponen los resultados del estudio, mostrando por cada columna los resultados para cada una de las tres tandas de pruebas. Cada celda representa el tiempo de retraso medido en esa prueba, representado en nanosegundos.

Medición	Ioctl	Procfs (escritura)	Procfs (lectura)
1	19968	199936	179968
2	50176	189952	170240
3	19968	179968	169984
4	19968	200192	219904
5	19968	240128	169728
6	19968	209920	159744
7	9984	179968	159744
8	9984	209920	159744
9	20224	179968	180224
10	9984	180224	169984
11	19968	199936	179712
12	9984	270080	160000
13	9984	199936	160000
14	9984	210176	170240
15	29952	209920	160256
16	9984	199936	179712
17	20224	199680	190208
18	20224	290048	250112
19	19968	200192	179968

Tabla B.1: Retraso medido en cada prueba (nanosegundos)

A simple vista puede observarse como los retrasos introducidos por *Procfs*, tanto en escritura como en lectura, son mucho mayores que los introducidos por *Ioctl*.

B.2 Análisis

Realizando un análisis sobre los resultados de las pruebas obtenemos la tabla B.2, en la que se representa para cada tanda de pruebas el retraso mínimo, el máximo y la media.

	Ioctl	Procfs (escritura)	Procfs (lectura)
Mínimo	9984	179968	159744
Máximo	50176	290048	250112
Media	**18445**	**207899**	**177341**

Tabla B.2: Retraso mínimo, máximo y medio para cada tipo (nanosegundos)

Como puede observarse, el tiempo introducido por *Ioctl* es aproximadamente diez veces menor que el introducido por *Procfs* en media. Este hecho motivó la decisión de emplear llamadas *Ioctl* en lugar de *Procfs* para la implementación del acceso a *RT-WMP*.

Apéndice C
Resultados de las pruebas

En éste apéndice se exponen los resultados de las pruebas de evaluación de rendimiento realizadas sobre el sistema implementado en este proyecto.

C.1 Ancho de banda

En la tabla C.1 se muestra un resumen de los resultados de las medidas de ancho de banda, mostrando los resultados según la plataforma y el número de nodos de la red. A su vez, se muestra el porcentaje de mejora que representa el resultado obtenido con *linux_ks* respecto de *linux_us*.

Nº Nodos	linux_ks	linux_us	% mejora
2	1588,71	1134,64	40,02
3	1177,97	828,25	42,22
4	830,70	697,34	19,12
5	606,62	537,75	12,81
6	430,02	406,35	5,82
7	363,11	341,17	6,43

Tabla C.1: Ancho de banda (Kbps) y porcentaje de mejora

Como puede observarse, la mejora va descendiendo conforma crece el tamaño de la red, algo que puede verse gráficamente en la figura C.1. Como ya se ha comentado en el capítulo 4 de la memoria, esto es debido a que excepto los dos ordenadores empleados para realizar las pruebas, el resto de nodos funcionan con MaRTE OS, con lo que ambos escenarios se asimilan conforme el tamaño de la red crece.

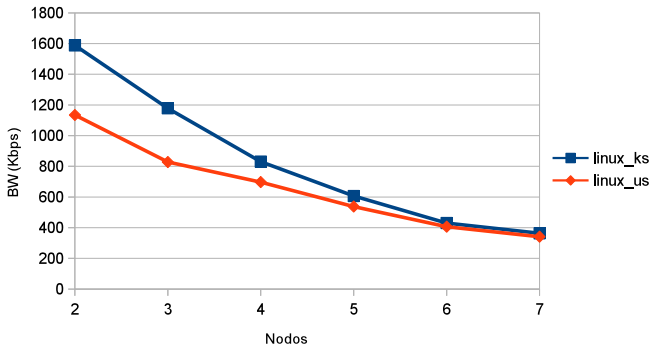


Figura C.1: **Ancho de banda**

C.2 Duración de una iteración

En esta prueba se estudió la duración de una iteración (tiempo entre el comienzo de una PAP hasta la siguiente) en el caso peor para la plataforma *linux_ks*.

En la tabla C.2 se muestran de los resultados de las medidas obtenidas con *wmpSniffer* para la duración de este intervalo en redes de diferente tamaño. Si todos los nodos ejecutasen sistemas operativos de tiempo real, la duración en el caso peor debería ser siempre la misma para redes del mismo tamaño, la calculada mediante la fórmula $(2N-3)t_t + (N-1)t_a + (N-1)t_m$, siendo N el número de nodos de la red, t_t el tiempo de transmisión de un token entre dos nodos, t_a el tiempo de transmisión de una autorización y t_m el tiempo de transmisión de un mensaje de máximo tamaño.

Sin embargo, Linux no lo es un sistema operativo de tiempo real, por lo que introduce indeterminismo temporal. Debido a ello, se muestra para cada tamaño el máximo y el mínimo tiempo medido para situaciones de caso peor, además de la media.

	WC LOOP Duration		
2	Media	Máximo	Mínimo
2	2171	9678	1812
3	4909	14636	4414
4	8558	14467	7567
5	12163	18552	10834
6	14494	15465	14227
7	18722	18734	18711

Tabla C.2: Duración del intervalo en el caso peor.

En la figura C.2 puede observarse la representación gráfica de la media para la duración de un intervalo en el caso peor de los diferentes tamaños de la red.

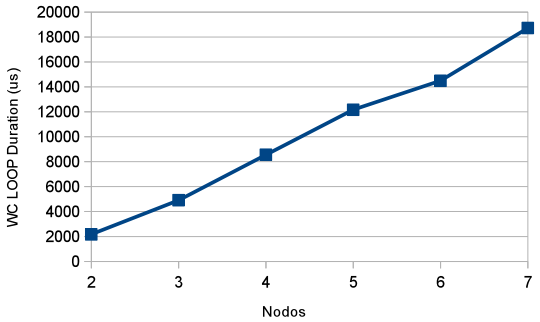


Figura C.2: **Duración de una iteración en el caso peor**

C.3 Resto de pruebas

Dado que los aspectos relacionados con el resto de pruebas realizadas ya han sido tratados en el capítulo 4, no volverán a repetirse en el presente apéndice.

Bibliografía

[1] Documentation extracted from the Linux kernel and mirrored on the web :: Kbuid. http://www.kernel.org/doc/Documentation/kbuild/modules.txt.

[2] MaRTE OS Home Page. `http://marte.unican.es/`.

[3] J. Corbet, A. Rubini, and G. Kroah-Hartman. *Linux Device Drivers, 3rd Edition.* O'Reilly Media, Feb. 2005.

[4] B. Nguyen. Linux Documentation Project - Linux Filesystem Hierarchy: 1.14. /proc. http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html.

[5] D. C. Plummer. An Ethernet Address Resolution Protocol. *Network Working Group - Request For Comments*, 826, November 1982.

[6] C. Sagues, A. Mosteo, D. Tardioli, J. Villarroel, L. Montano, and L. Montano. Sistema multi-robot en localización e identificación de vehículos. *Revista Iberoamericana de Automática e Informática*, 2011.

[7] P. J. Salzman, M. Burian, and O. Pomerantz. The Linux Kernel Module Programming Guide. http://tldp.org/LDP/lkmpg/2.6/html/.

[8] E. Schrock. Reflections on OS integration: A brief history of /proc. http://blogs.oracle.com/eschrock/entry/the_power_of_proc/, June 2004.

[9] D. Sicignano, D. Tardioli, S. Cabrero, and J. L. Villarroel. Real-Time Wireless Multi-Hop protocol in Underground Voice Communication. *Ad Hoc Networks*, 2011.

[10] D. Sicignano, D. Tardioli, and J. L. Villarroel. QoS over Real-Time Wireless Multi-hop Protocol. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 28:110–128, 2010.

[11] W. R. Stevens. *Advanced Programming in the UNIX Environment.* Addison-Welsey, 1992. Section 3.14.

[12] D. Tardioli. Real-Time Communication in Wireless ad-hoc networks. The RT-WMP protocol. *Universidad de Zaragoza*, October 2010.

[13] D. Tardioli, A. R. Mosteo, L. Riazuelo, J. L. Villarroel, and L. Montano. Enforcing Network Connectivity in Robot Team Missions. *The International Journal of Robotics Research. doi:10.1177/0278364909358274*, Vol. 29, No.4:460–480, April 2010.

[14] D. Tardioli and J. L. Villarroel. Real Time Communications over 802.11: RT-WMP. *IEEE International Conference on Mobile Adhoc and Sensor Systems, 2007. MASS 2007.*, pages 1–11, Oct. 2007.

Índice de figuras

Índice de tablas

Acrónimos y siglas

ARP	Protocolo de resolución de direcciones (Address Resolution Protocol)
ATP	Fase de autorización de la transmisión (Authorization Transmission Phase)
FIFO	Primero en entrar, primero en salir (First in, first out)
FPU	Unidad de coma flotante (Floating-point unit)
ICMP	Protocolo de Mensajes de Control de Internet (Internet Control Message Protocol)
Ioctl	Input/Output control
IP	Protocolo de Internet (Internet protocol)
LKM	Módulo del kernel de Linux (Linux Loadable Kernel Module)
LQM	Matriz de calidad de enlace (Link Quality Matrix)
LNL	Capa de red de Linux (Linux network layer)
MANET	Red móvil ad-ho (Mobile ad-hoc network)
MPM	Mensaje más prioritario (More Priority Message)
MTP	Fase de transmisión del mensaje (Message Transmision Phase)
MTU	Unidad máxima de transferencia (Maximum Transfer Unit)
P2P	Red entre pares o red punto a punto (Peer to peer)
PAP	Fase de arbitraje de la prioridad (Priority Arbitration Phase)
Procfs	Sistema de ficheros de procesos (Process filesystem)
QoS	Calidad de servicio (Quality of Service)
RT-WMP	Real-Time Wireless Multi-hop Protocol
TCP	Protocolo de control de transmisión (Transmission Control Protocol)
UDP	Protocolo de datagramas de usuario (User Datagram Protocol)
VoIP	Voz sobre IP (Voice over IP)