Universidad
Rey Juan Carlos

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Curso Académico 2007/2008

Proyecto Fin de Carrera

MONITORIZACIÓN DE PAQUETES IP EN ENTORNOS NetGUI

Autor: Enrique Gómez Monreal
Tutor: Eva María Castro Barbero

Agradecimientos

Quiero dedicar este trabajo a mi familia, y en especial a mis padres que siempre se preocupan por mí y me han apoyado en todo cuanto he emprendido.

También quiero agradecer a Eva María Castro su ayuda y paciencia en el desarrollo de este trabajo, así como la confianza que ha depositado en mí, ya que sin su apoyo este proyecto no hubiera sido posible.

Resumen

Todo proyecto que nos planteamos en la vida intenta solucionar un problema o llenar un vacío. Por ello, una de las características más deseables que necesitan las redes de ordenadores es poder realizar su estudio desde un punto de vista más *práctico* de lo que actualmente se realiza, y éste es el vacío que con este proyecto se pretende llenar.

El objetivo de este proyecto es obtener una monitorización de los datagramas que viajan en la red, en la que se muestre el orden de los paquetes y el camino que han seguido dichos paquetes a través de todos los nodos de la red a nivel IP, de modo que facilite el estudio del encaminamiento en los escenarios de red.

Actualmente existen herramientas que nos ofrecen la posibilidad de estudiar las redes, sin embargo sus funcionalidades se centran en el estudio de lo que ocurre en una máquina, en vez de lo que ocurre en el conjunto de una red de ordenadores.

Por todo esto, se han realizado modificaciones en el protocolo **IP** para añadir información extra a los datagramas IP, de modo que finalmente nos puedan servir de ayuda para que se observe el recorrido que ha seguido cada uno de los datagramas en la red.

Además se han realizado pruebas en distintos escenarios de red en las que se han obtenido resultados que nos acercan más a poder simplificar el estudio de las redes basadas en los protocolos TCP/IP.

Por tanto la finalidad de este proyecto no es otra que la de proporcionar a los estudiantes una herramienta que les facilite la comprensión de las redes de ordenadores.

Índice general

Índice de figuras

Capítulo 1

Introducción

Las redes de ordenadores se están convirtiendo en un herramienta cada día más familiar y conocida, que usamos a diario sin apenas ser conscientes de ello. La mayoría de las veces se utilizan solicitando servicios de Internet; un ejemplo puede ser cada vez que accedemos a una página web o comprobamos el correo electrónico. El estudio de estas redes tiende a ser muy teórico, y se echa en falta que se pudiera estudiar desde un aspecto más **práctico** puesto que ayudaría a comprender mejor el funcionamiento de las redes.

El mayor inconveniente para poner realizar este estudio de un modo práctico, es la necesidad de que existan varios dispositivos hardware *(routers, hubs, switches, PCs, etc.)* conectados para poder configurar distintos escenarios o topologías de red. De esa manera se podría observar el intercambio de mensajes que se suceden en la red entre los distintos dispositivos. Evidentemente no siempre se puede contar con dicho hardware, y por tanto se reducen las posibilidades del estudio práctico de redes.

Existen herramientas que se usan para analizar, emular y simular redes de ordenadores y que por tanto, se pueden usar de forma didáctica para el estudio de las redes. Sin embargo, aún falta perfeccionar mucho estas herramientas para que sean fáciles de utilizar por cualquier persona con unos conocimientos básicos.

En este capítulo se describirán las herramientas de emulación que se han utilizado para desarrollar el trabajo de este proyecto fin de carrera.

1.1. Entornos de emulación para el estudio de redes de ordenadores

Las ventajas que ofrecen las herramientas de emulación de redes son variadas. El trabajo con ellas implica un bajo coste con respecto a redes físicas, permitiendo la posibilidad de realizar múltiples configuraciones para laboratorios de un modo sencillo. Además nos permiten analizar de forma controlada todos los eventos que ocurren en una red de ordenadores, así como el comportamiento de las máquinas que intervienen. [2]

Para el desarrollo de este proyecto fin de carrera se ha utilizado el software para la emulación de redes de ordenadores, *UML*, *Netkit* y *NetGUI*, que se describe en las siguientes secciones.

1.1.1. UML

UML, acrónimo de **User Mode Linux**, es una modificación del núcleo del sistema operativo GNU/Linux para que funcione sobre su propia interfaz de llamadas al sistema. De este modo, el kernel[1] puede ser arrancado para operar como un proceso de usuario más en una máquina que tenga instalado GNU/Linux *(Máquina host o anfitrión)*. A cada una de los procesos UML que emula un ordenador o un router se les suele llamar **máquinas virtuales**.

UML proporciona la creación de máquinas virtuales que incluso podrían disponer de más componentes (virtuales) hardware y software, que la propia máquina física que las sostiene. El sistema de ficheros de las máquinas virtuales se almacenan en un único fichero por máquina virtual, en la máquina física. La ventaja de las máquinas virtuales es que cualquier cambio o daño que sufran las máquinas virtuales, no se ve reflejado en la máquina física.

El incoveniente que presenta esta herramienta es que para configurar un entorno de red son necesarios conocimientos avanzados sobre el sistema.

[1] En informática, el núcleo (también conocido en español por el anglicismo *kernel*) es la parte fundamental de un sistema operativo. Es el software responsable de facilitar a los distintos programas acceso seguro al hardware de la computadora o en forma más básica, es el encargado de gestionar recursos, a través de servicios de llamada al sistema.

1.1.2. Netkit

Desarrollado por la Universidad de Roma, Netkit[2] fue concebido para ofrecer un entorno virtual donde se pueda configurar y realizar pruebas experimentales de topologías de red con un bajo coste y con poco esfuerzo. Permite la posibilidad de crear virtualmente dispositivos de red *(routers, switches, ordenadores, etc)* y que pueden ser fácilmente interconectados para finalmente crear redes usando una única máquina física.

Netkit es software libre, bajo licencia GPL y está basado en UML. El propósito de este proyecto es solventar las dificultades que se pueden encontrar al establecer entornos de red usando UML.

1.2. NetGUI

NetGUI[3] es una herramienta especialmente diseñada para ser usada por estudiantes de primer curso de ingenierías de telecomunicaciones. Se entiende que estos estudiantes podrían tener poca o ninguna experiencia con entornos de línea de comandos como la shell de UNIX. Aunque ya existían entornos como Netkit para poder trabajar con redes emuladas, aún era necesaria una herramienta para poder crear topologías de red de forma más sencilla sin tener que recurrir a la línea de comandos.

NetGUI ofrece una interfaz gráfica para Netkit, en la que se puede encontrar una barra de herramientas con una serie de botones (Figura 1.1). El usuario puede dibujar topologías de red usando los elementos que se ofrecen en la barra de herramientas, y una vez diseñada puede arrancar las máquinas virtuales. Mientras se están ejecutando estas máquinas, el usuario puede interactuar con las consolas de éstas.

En la figura 1.2 se puede observar gráficamente la arquitectura sobre la que se apoyan cada una de las implementaciones mencionadas anteriormente.

1.3. Contenido de la memoria

La organización de esta memoria es la que se expone a continuación.

Después de esta breve introducción, en la que partiendo de una visión global del mundo de las redes de ordenadores, nos hemos sumergido en las particularidades que

[2] http://www.netkit.org
[3] http://mobiquo.dat.escet.urjc.es/netgui

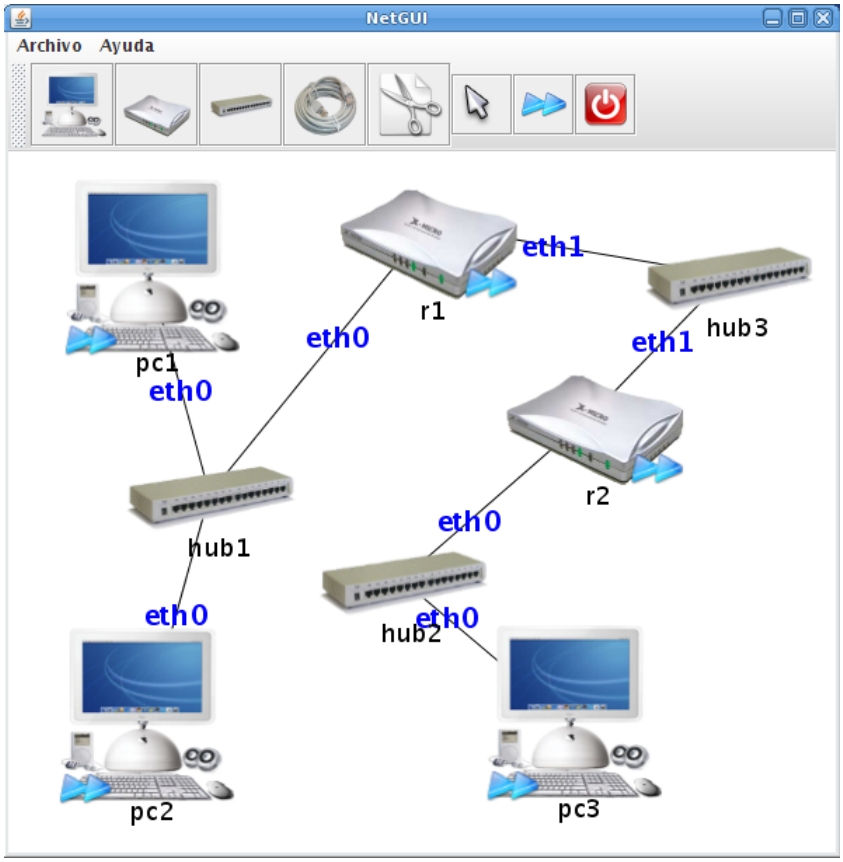


Figura 1.1: *Intefaz de NetGUI.*

atañen a este proyecto, continuaremos describiendo las necesidades que han llevado a la realización de este proyecto fin de carrera. Todo esto se presenta en el capítulo "Objetivos" (capítulo 2).

En el capítulo 3, se describe de manera detallada la metodología empleada y los pasos seguidos pretendiendo dar una definición completa y global de la funcionalidad de operación que va a tener disponible la aplicación.

Avanzando un poco más en el documento, nos encontramos con la validación del proyecto, explicada en el capítulo 4 donde se describen las pruebas que confirman el comportamiento que se especificó.

Todo esto se completa con el capítulo 5 de conclusiones, donde se expone la contribución que este proyecto implica para el estudio de las redes y las posible mejoras futuras que se podrían aplicar.

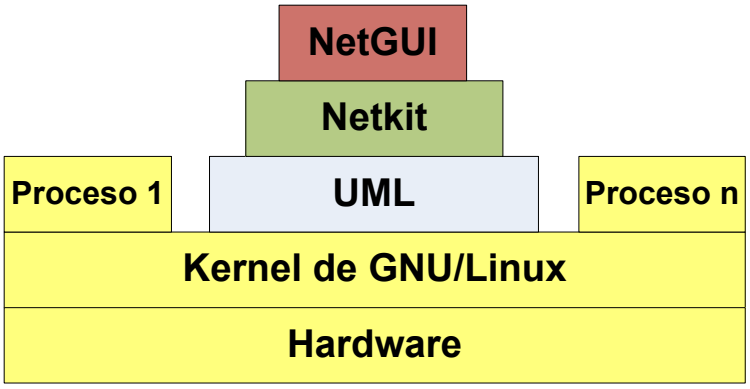


Figura 1.2: *Arquitectura básisca UML.*

Y por último, se encuentra la bibliografía en la cual se detalla la documentación utilizada para el desarrollo de esta proyecto fin de carrera.

Capítulo 2

Objetivos

Una vez presentado en el capítulo 1 el contexto general y particular de este proyecto, vamos a fijar los objetivos concretos de este proyecto fin de carrera y los requisitos que han condicionado su desarrollo.

2.1. Descripción del problema

El estudio de lo que ocurre en una red de ordenadores es un tema complejo. Una simple descarga de una página web que esté alojada en un sistema remoto requiere el intercambio de varios mensajes entre diferentes máquinas para que eso sea posible. Por este motivo necesitamos herramientas que puedan servir de soporte al estudio del funcionamiento de las redes de ordenadores.

El objetivo principal de este proyecto es monitorizar todos los datagramas IP generados en una red de ordenadores para conocer qué máquinas han intercambiado mensajes y el orden en el cuál han sido intercambiados.

Hay formas de hacer esta monitorización a partir de capturas de herramientas como TCPDump o Wireshark, que permiten analizar el tráfico que circula por la red. Sin embargo, no hay forma de ordenar esos mensajes, ya que los relojes de las máquinas virtuales no están sincronizados.

Como finalidad, se desea introducir información adicional en los paquetes IP que nos permita realizar un seguimiento del camino que va llevando ese paquete a lo largo de la red hasta llegar al destino. Para ello, cada nodo del camino almacenará información de los paquetes que procese y al finalizar la monitorización esa información será procesada por un nodo para presentar de forma organizada qué datagramas IP se han generado y qué camino han seguido hasta llegar a su destino.

2.1.1. Subobjectivos del proyecto

Este proyecto se ha articulado en los siguientes cuatro subobjetivos:

- Para empezar hay que definir qué tipo de información adicional viajará en los datagramas IP. Se debe tener en cuenta que la finalidad del proyecto es que los datagramas del sistema se puedan ordenar. Por ello, la información adicional que viaje nos deberá permitir definir un orden cronológico de todos los datagramas que viajen en la red monitorizada.
- A continuación, se debe de modificar el kernel de UML para que los paquetes IP incluyan dicha información y se propague por todos los nodos hasta llegar al destino. En esta parte habrá que encargarse de que cada vez que se genera un nuevo datagrama IP, automáticamente se añadan los datos pertinentes al datagrama, y garantizar que cada nodo que lo propaga, modifique convenientemente dicha información.
- Modificar el kernel de UML para que cada vez que un nodo procese un datagrama IP, se almacene la información relacionada con NetGUI de todos los paquetes que se están procesando. Esto supondrá que cada vez que se vaya a *enviar*, *recibir* o *reenviar* un paquete de este tipo, se añada nueva información al *log*, con los datos asociados.
- Como último paso, se tienen que recoger los datos que han almacenado todos los nodos de la red, para posteriormente procesarlos y poder ofrecerlos de forma ordenada. Para ello se utilizará un nodo que llamaremos *Monitor de NetGUI*, que recibirá toda esta información y la procesará adecuadamente.

2.2. Requisitos

Los requisitos necesarios para la realización de este proyecto son los siguientes:

1. Necesidad de conocimientos en Lenguaje C, para desarrollar las nuevas funcionalidades en el Kernel al ser el entorno en el que está implementado.
2. Conocimientos de la pila de protocolos TCP/IP para comprender el funcionamiento de las redes y poder desarrollar las nuevas funcionalidades de acuerdo con la estructura del protocolo.

3. Utilización de la plataforma de máquinas virtuales que ofrecen Netkit junto con UML para poder realizar las pruebas con la modificaciones del Kernel del sistema operativo GNU Linux desarrolladas en este PFC. Nos facilita mucho la tarea el poder utilizar entornos virtuales para comprobar el correcto funcionamiento de las modificaciones realizadas en el núcleo (Kernel).

4. Utilización de la herramienta **NetGUI** que agilizará la tarea de establecer escenarios que permitirán realizar las pruebas del sistema de forma sencilla.

2.3. Metodología y plan de trabajo

El plan de trabajo que se ha seguido para la realización de este proyecto sigue el *modelo de desarrollo en espiral basado en prototipos*. La elección de este modelo de desarrollo se basa en la necesidad de separar el comportamiento final en varias subtareas más sencillas que conjuntamente compondrán el comportamiento final del sistema. La ventaja de utilizar este modelo de desarrollo es la flexibilidad en cuanto a posibles cambios de requisitos.

Figura 2.1: *Modelo de desarrollo en espiral.*

Este modelo de desarrollo se caracteriza por la relación de las subtareas en un número determinado de ciclos. En cada uno de estos ciclos existen cuatro etapas:

Análisis de requisitos, Diseño e implementación, Pruebas y Planificación del próximo ciclo de desarrollo. (Ver ilustración 2.1).

Para la elaboración de este proyecto se han realizado los siguientes ciclos:

1. Etapa de formación y familiarización con la *infraestructura* y *software* utilizado.
2. Estudio de pila de protocolos TCP/IP y de su implementación en el Kernel de GNU Linux.
3. Análisis, diseño, implementación y pruebas de las opciones IP para NetGUI.
4. Análisis, diseño, implementación y pruebas del algoritmo de ordenación de los datos gestionados en el *Monitor de NetGUI.*
5. Implementación de la interfaz gráfica para la presentación de los resultados.
6. Pruebas para comprobar el funcionamiento de las modificaciones en el kernel del sistema operativo y del Monitor de NetGUI.
7. Escritura de la memoria.

Capítulo 3

Diseño e Implementación

La idea de este proyecto reside en añadir información nueva en los paquetes IP sin variar el formato del paquete, de modo que cada máquina que intervenga en el proceso de transmisión procese, y modifique dicha información convenientemente. Además, cada máquina, guardará en un fichero todos estos datos para posteriormente enviarlos a una máquina que llamaremos **monitor de NetGUI**.

Esa máquina, se encargará de recibir la información de todas las máquinas que hayan intervenido, y posteriormente reorganizará los datos para poder mostrar un informe detallado de los nodos por los que ha pasado cada uno de los paquetes.

En la figura 3.1 se ha diseñado un esquema básico en el que aparecen varias máquinas y el **monitor de NetGUI**. Tanto los routers, como los PCs que aparecen en el esquema enviarán información de todos los paquetes IP que *reciben*, *envían* o *reenvían* a la máquina monitor de NetGUI, que en la red sólamente se dedicará a esperar las conexiones pertinentes de cada máquina para poder recibir la información del resto de los nodos, y finalmente procesarla.

3.1. Paquete IP y campo de opciones

La cabecera básica de un paquete IP consta de tan sólo 20 bytes, sin embargo, ésta puede incrementarse hasta llegar a un tamaño máximo de 60 bytes. Así permite que el campo de las opciones del datagrama IP pueda tener un máximo de 40 bytes.

La mayoría de las opciones de IP se usan en contextos muy particulares. Varias opciones se pueden combinar en un mismo paquete IP, pero a excepción de las opciones "*Final de lista de opciones*" y "*Ninguna operación*", puede haber como mucho una instancia de cada una en la cabecera. La presencia de opciones puede influenciar en

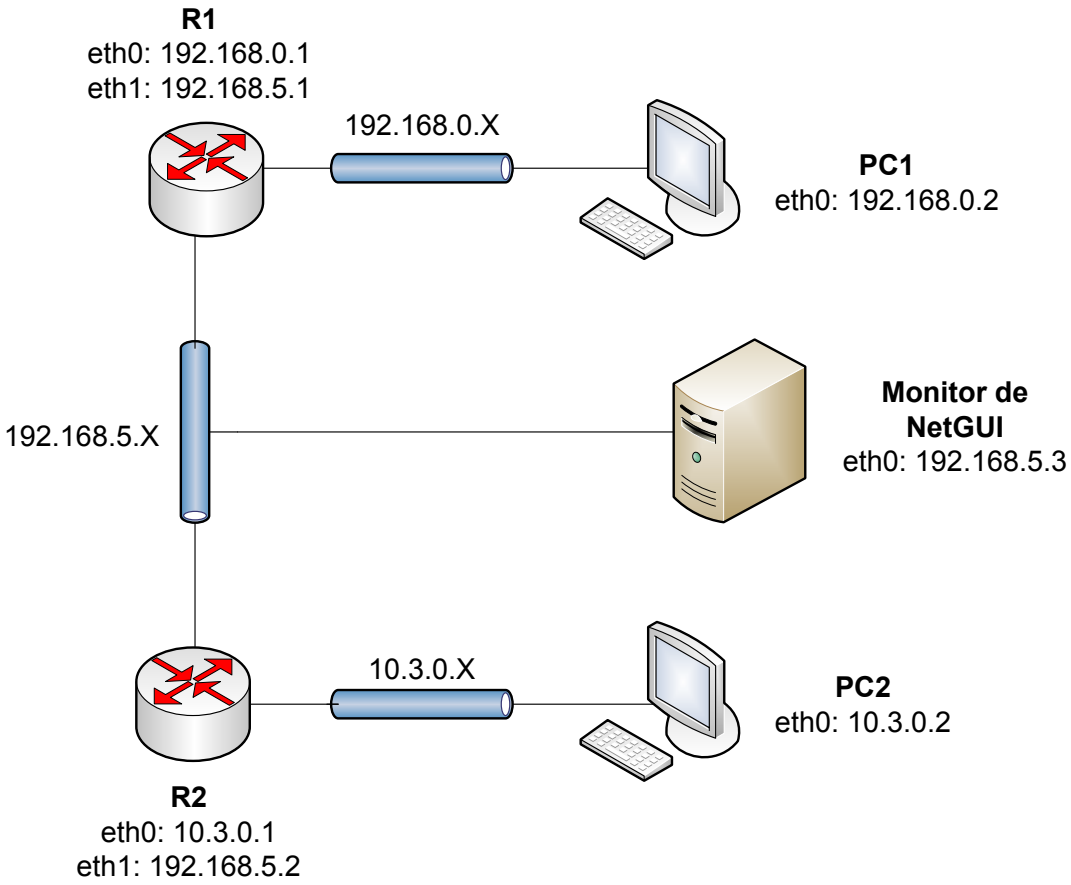


Figura 3.1: *Topología de red ejemplo en la que se puede observar el* ***Monitor de NetGUI****.*

el proceso de fragmentación y defragmentación.

No es común el uso de este campo, ya que producen sobrecarga en los routers debido al tiempo que se tarda en procesar estas opciones, así que habitualmente no se usan. Sin embargo, no nos preocupa la posible sobrecarga de los routers al procesar las opciones ya que esta aplicación está destinada para ser usada en máquinas virtuales para entornos meramente didácticos con el objetivo de mostrar de forma más sencilla el funcionamiento de las redes de ordenadores.

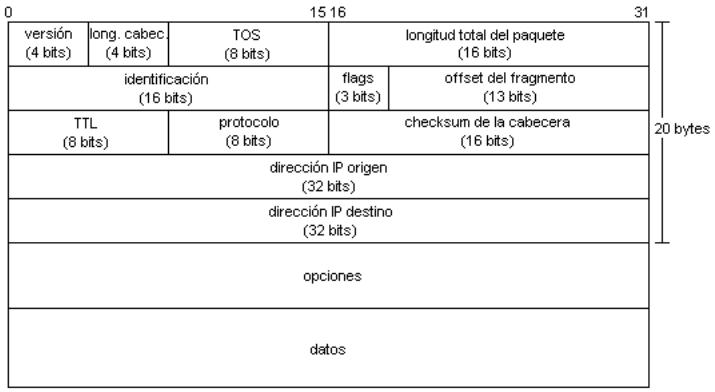


Figura 3.2: *Campos de un datagrama IP.*

3.1.1. Formato del campo de las opciones de IP

Algunas opciones son muy sencillas y pueden ser especificadas en un sólo byte u octecto; existen opciones más complejas que requieren un formato más flexible (véase la figura 3.3).

Teniendo esto en cuenta, existen dos posibles formatos para el campo de las opciones.

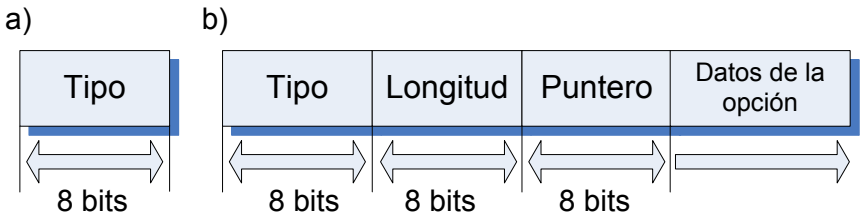


Figura 3.3: *(a) Formato de la opción simple de IP; (b) Formato de la opción compuesta de IP.*

Formato de opciones simple

Se determina con un sólo octeto indicando el **tipo de opción**, el cual está dividido en tres campos.

- **Indicador de copia**: 1 bit. En caso de fragmentación, la opción se copiará o no a cada nuevo fragmento según el valor de este campo.
  - 0 - No se copia.
  - 1 - Se copia.
- **Clase de opción**: 2 bits. Las posibles clases son:
  - 00 - Control.
  - 01 - Reservada.
  - 10 - Depuración y mediciones.
  - 11 - Reservada.
- **Número de opción**: 5 bits. Identificador de la opción.

Formato de opciones compuesto

Se compone de un octecto para el **Tipo de opción** (como en el caso del formato de opciones simple), otro para el **Tamaño de opción**, y uno o más octetos conformando los **Datos de opción**.

El cuadro 3.1 muestra las opciones actualmente definidas.

Símbolo usado en el Kernel	Clase	Número	Copia	Tamaño	Descripción
IPOPT_END	00	0	0	-	Final de lista de opciones. Formato simple.
IPOPT_NOOP	00	1	0	-	Ninguna operacion (NOP). Formato simple.
IPOPT_SEC	00	2	1	11	Seguridad.
IPOPT_LSRR	00	3	1	variable	Enrutado desde el Origen, abierto (Loose Source Routing).
IPOPT_SSRR	00	9	1	variable	Enrutado desde el Origen, estricto (Strict Source Routing).
IPOPT_RR	00	7	0	variable	Registro de Ruta (Record Route).
IPOPT_SID	00	8	1	4	Identificador de flujo (Stream ID).
IPOPT_TIMESTAMP	11	4	0	variable	Marca de tiempo (Internet Timestamping).

Cuadro 3.1: Valores definidos para el formato de opciones compuesto de IP.

3.1.2. Formato de las opciones de NetGUI

En nuestro caso, vamos a definir una nueva opción, que pueda almacenar la siguiente información:

- **Identificador del paquete**. Se trata de un número que identificará unívocamente cada uno de los paquetes que envía una máquina. Este identificador lo establece el nodo origen, y no se verá modificado por ninguna de las máquinas que procese el paquete. De esta manera, el nodo destino recibirá este identificador sin ningún cambio desde que lo envió el nodo origen. Este valor se implementa como un contador generado por el emisor y servirá posteriormente para ordenar los diferentes paquetes IP que envía una determinada máquina.

  Sin embargo, nuestro objetivo es que al final se puedan ordenar las operaciones de envío, reenvío y recepción de paquetes IP que se ejecutan en un conjunto de máquinas para permitir la transmisión de información entre la máquina origen y su destino, y este identificador por sí solo no es suficiente.

  Estas operaciones podrían ordenarse si los relojes de las máquinas virtuales estuvieran sincronizados. Desafortunadamente, no es así y en este proyecto fin de carrera se han implementado otros mecanismos para permitir la ordenación de los mismos.

- **Identificador de reloj de Lamport**. Debido a la necesidad de ordenar las operaciones de envío, reenvío y recepción de paquetes IP, se optó por utilizar **Relojes Lógicos**, como el reloj de Lamport, que formará parte de la información que se incluye en la cabecera IP.

  Leslie Lamport, ingeniero informático estadounidense, en un artículo clásico, Lamport (1978), demostró que la sincronización de relojes es posible y presentó un algoritmo para lograr esto.

  Cada máquina posee un circuito interno en el que registra su reloj local. A la hora de sincronizar sistemas distribuidos este reloj no es válido ya que es imposible garantizar que relojes de distintas máquinas oscilen con la misma frecuencia.

  El *algoritmo de Lamport* [4] proporciona un ordenamiento de eventos sin ambigüedad basado en que los valores de tiempo asignados a los eventos no tienen porqué ser cercanos a los tiempos reales en los que ocurren. Un reloj lógico de Lamport es un contador software que se incrementa monótamente, cuyos valores no necesitan tener ninguna relación particular con ningún reloj físico.

  Para sincronizar los relojes lógicos, **Lamport** definió la siguiente relación:

- Si a y b son eventos en el mismo proceso y a ocurre antes de b, entonces $a \rightarrow b$ es verdadero.
- *"Ocurre antes de"* es una relación transitiva: Si $a \rightarrow b$ y $b \rightarrow c$, entonces $a \rightarrow c$.
- Si dos eventos x e y están en procesos diferentes que no intercambian mensajes, entonces $x \rightarrow y$ no es verdadero, pero tampoco lo es $y \rightarrow x$. Se dice que son eventos concurrentes.

Ahora necesitamos una forma de medir el tiempo tal que a cada evento a, le podamos asociar un valor del tiempo *C(a)* en el que todos los procesos estén de acuerdo. Se debe cumplir que:

- Si $a \rightarrow b$ entonces $C(a) < C(b)$.
- El tiempo del reloj C, siempre debe de ser creciente y nunca decreciente.

Con este algoritmo se puede asignar un tiempo a todos los eventos en un *sistema distribuido*, con las siguientes condiciones:

- Si a ocurre antes de b en el mismo proceso, $C(a) < C(b)$.
- Si a y b son en envío y recepción de un mensaje, $C(a) < C(b)$.
- Para todos los eventos relacionados a y b, *C(a)* es distinto de *C(b)*.

En la figura 3.4 se puede apreciar un sistema que no está sincronizado *(Esquema de la izquierda)*, y otro que está sincronizado con el algoritmo de los relojes de Lamport *(Esquema de la derecha)*.

Este algoritmo es una forma de obtener un orden total de todos los eventos del sistema.

En el sistema GNU Linux pueden darse situaciones en las que haya dos o más procesos o hilos *(threads)* que intentan acceder a un recurso común (escribir un fichero, leer una zona de memoria ...). El problema existe si estas operaciones se hacen simultáneamente y no hay un control sobre ellas, ya que se podrían generar inconsistencias en los recursos.

Aplicándolo a nuestro sistema, podría darse un escenario en el que una máquina recibiera y enviara multitud de paquetes a la vez (algo por otro lado es bastante común), y que por ello varios hilos intenten acceder simultáneamente al **identificador de Lamport** y al **identificador del paquete** que genera una determinada máquina. Para evitar que se produzcan inconsistencias, hemos utilizado unas variables que soportan operaciones **atómicas** para implementar estos identificadores.

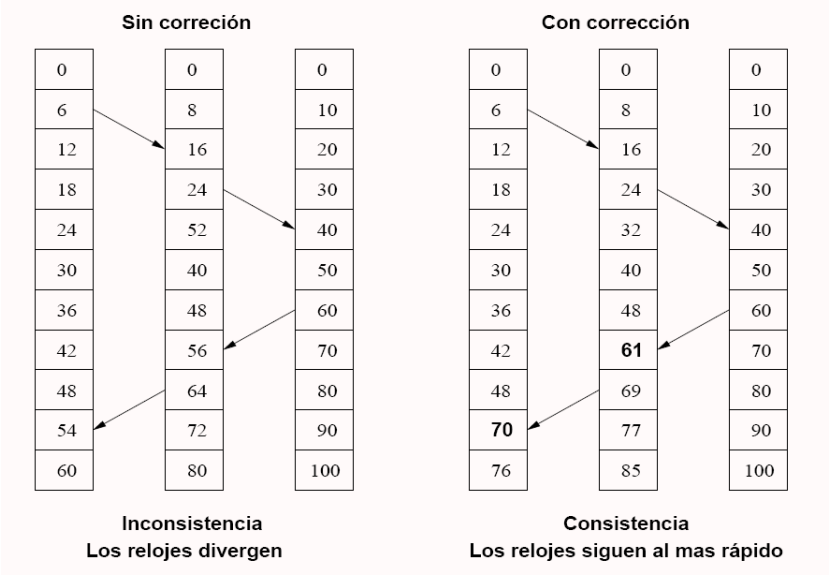


Figura 3.4: *Ejemplo de sincronización con relojes de Lamport.*

Estas variables son del tipo *atomic_t* y se utilizan sobretodo para realizar operaciones aritméticas atómicas. Con ellas, garantizamos que hasta que un hilo no haya realizado su operación con esa variable, otro hilo no podrá acceder a ella. Además, cuentan con funciones atómicas de lectura *(atomic_read())*, de escritura *(atomic_set())*, y nosotros utilizaremos también una operación de incremento *(atomic_inc())*, que nos facilitará la labor a la hora de manejar los identificadores de NetGUI.

Para implementar la nueva opción, necesitaremos un formato de opciones **compuesto**. Ambos identificadores se han definido con un tipo de datos *atomic_t*, que es un tipo *int* del lenguaje C[1], con la característica de atomicidad. Estas variables tienen un tamaño de 32 bits (4 bytes). Usaremos el campo de **Tipo de opción** (1 byte), más el campo de **Tamaño de opción** (1 byte) más los 8 bytes que suman ambos identificadores. En total las opciones de NetGUI tendrán un tamaño de **10 bytes**. Cabe destacar que el número de bytes del campo de las opciones tiene que ser múltiplo de 4. Por ello, se reservan **12 bytes** para las opciones de NetGUI. [3]

En la figura 3.5 se muestra un esquema en el que se definen con exactitud los bytes utilizados por cada campo de la opción.

Esta estructura se encuentra en el fichero *include/linux/ip.h*.

[1]El lenguaje C es el utilizado mayoritariamente en el código fuente del Kernel de GNU Linux.

Símbolo usado en el Kernel	Clase	Número	Copia	Tamaño
IPOPT_NETGUI	00	31	0	10

Cuadro 3.2: Valores de los subcampos de la opción IP de NetGUI.

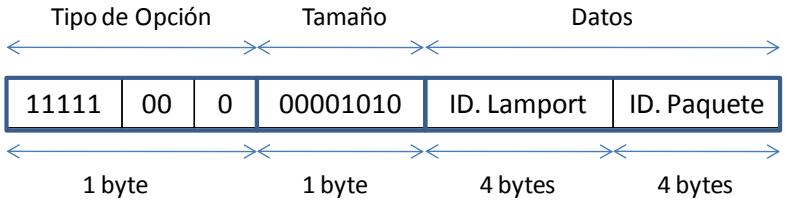


Figura 3.5: *Estructura de las opciones de NetGUI.*

```
1  #define IPOPT_END        (0 |IPOPT_CONTROL)
2  #define IPOPT_NOOP       (1 |IPOPT_CONTROL)
3  #define IPOPT_SEC        (2 |IPOPT_CONTROL|IPOPT_COPY)
4  #define IPOPT_LSRR       (3 |IPOPT_CONTROL|IPOPT_COPY)
5  #define IPOPT_TIMESTAMP  (4 |IPOPT_MEASUREMENT)
6  #define IPOPT_CIPSO      (6 |IPOPT_CONTROL|IPOPT_COPY)
7  #define IPOPT_RR         (7 |IPOPT_CONTROL)
8  #define IPOPT_SID        (8 |IPOPT_CONTROL|IPOPT_COPY)
9  #define IPOPT_SSRR       (9 |IPOPT_CONTROL|IPOPT_COPY)
10 #define IPOPT_RA         (20|IPOPT_CONTROL|IPOPT_COPY)
11 #define IPOPT_NETGUI     (31|IPOPT_CONTROL)
```

Listado 3.1: Definición de IPOPT_NETGUI.

Como se puede apreciar en el listado de código 3.1, **IPOPT_NETGUI** se ha definido como una opción de tipo de control, y no se ha establecido el *flag IPOPT_COPY* ya que no es necesario que los datos se copien en todos los fragmentos de un paquete IP. El flag 31 (11111 en binario) corresponde con el campo que indica el número de opción. Con este número nos resultará sencillo identificar un paquete que contenga la opción de NetGUI.

3.1.3. Formato del log de NetGUI

Cada nodo que procese un paquete que lleve la opción de NetGUI debe de registrarlo de algún modo en la máquina. Más tarde esa información se debe de enviar a la máquina *monitor de NetGUI*, que recibirá la información de todas las máquinas de la red.

En los sistemas GNU Linux, existe un fichero de log en */var/log/messages* que se utiliza para registrar los eventos que ocurren en el sistema. No se trata de un log de *"alertas"*, si no que es de carácter informativo.

Hemos decidido utilizar el fichero */var/log/messages* para almacenar mensajes relacionados con los eventos que ocurren a nivel IP en una máquina, así pues, cada vez que se *envíe*, *reenvie* o *reciba* un paquete con la opción de NetGUI, se registrará en este fichero toda la información relacionada con esos eventos. De este modo, se podrá acceder fácilmente a la información de NetGUI en cada máquina.

El formato de cada línea de NetGUI en este log será:

NETGUI, TCP/UDP, SEND/RECV/FRWD, Id.Lamport, Id.Paquete, IPOrigen:PuertoOrigen, IPDestino:PuertoDestino, TTL

Además, el kernel automáticamente añade en cada línea del log la fecha en la que se ha registrado la línea con una precisión de segundos, y el nombre de la máquina o *hostname*.

Un ejemplo de una línea sería este:

May 29 12:55:05 pc2 kernel: NETGUI, TCP, SEND, 277, 53, 10.3.0.2:47536, 192.168.0.3:5555, 64

Esta línea significa que el *29 de mayo* a las *12:55:05* el kernel de la máquina *pc2* ha ejecutado una operación de *envío* de un segmento *TCP* desde la máquina *10.3.0.2* y puerto *47536* hasta la máquina *192.168.0.3* y puerto *5555* con valor de TTL *64*, y los siguientes campos en las opciones de **NetGUI**: *277* para el identificador de Lamport, y *53* para el identificador del paquete.

Estos datos de cada máquina serán suficientes para poder realizar un informe completo sobre el tráfico IP que se genere en un escenario de red determinado.

3.2. Enviar los datos de NetGUI

Una vez que ya tenemos definida la estructura de la opción, hay que añadirla en cada paquete IP que se envíe. Según la lógica del Kernel de GNU Linux, las opciones de los paquetes sólo se pueden establecer a nivel de aplicación (L5) en la pila de protocolos TCP/IP y para ello ya existe la llamada al sistema *setsockopt()*. Sin embargo, en nuestro esquema hemos roto con esta lógica, e introducimos las opciones en el nivel de la capa de red, **IP** (L3).

La función *ip_options_get_alloc_netgui()* (Listado 3.2) se encargará de reservar memoria para añadir las opciones de NetGUI en los paquetes IP que no tienen otras opciones. Se encuentra en el fichero *net/ipv4/ip_options.c*.

```
1 struct ip_options *ip_options_get_alloc_netgui()
2 {
3         struct ip_options *opt = ip_options_get_alloc(netgui_size);
4         return opt;
5 }
```

Listado 3.2: Función que reserva memoria para el campo de las opciones de IP.

En el listado 3.3 se muestra la función que añadirá la opción de netkit (IPOPT_NETGUI) a una estructura del tipo *struct ip_options*. De modo que dicha estructura más tarde se añadirá al paquete IP. Esta función se encuentra en el fichero *net/ipv4/ip_output.c*.

```
static void fill_netgui_opt(struct ip_options *opt){

        unsigned char lOpt = netgui_size;
        int id = htonl(atomic_read(&netgui_lamport_id));
        int counter = htonl(atomic_read(&netgui_msg_counter));
        unsigned char *aux;

        atomic_inc(&netgui_lamport_id);
        atomic_inc(&netgui_msg_counter);

        aux = opt->__data;

        opt->__data[0]=IPOPT_NETGUI;
        aux+=1;
        memcpy(aux,&lOpt,sizeof(lOpt));
        aux+=sizeof(lOpt);
        memcpy(aux,&id,sizeof(id));
        aux+=sizeof(id);
        memcpy(aux,&counter,sizeof(counter));

        while (lOpt & 3)
                opt->__data[lOpt++] = IPOPT_END;

        opt->optlen = lOpt;
        opt->is_data = 1;
}
```

Listado 3.3: Función para rellenar el campo de opciones.

Y finalmente tenemos una función en el fichero *net/ipv4/ip_output.c* que invoca a las dos definidas anteriormente, véase el listado 3.4. Es decir, primero reserva la memoria necesaria para almacenar la opción de NetGUI, para posteriormente rellenarla con el valor de dicha opción.

```
static struct ip_options *create_netgui_opt(){

        struct ip_options *opt = ip_options_get_alloc_netgui();
        fill_netgui_opt(opt);

        return opt;
}
```

Listado 3.4: Función que añade el campo de opciones de NetGUI.

3.2.1. Añadir la opción de NetGUI a los paquetes IP

El siguiente paso de desarrollo consiste en buscar el lugar apropiado del Kernel para introducir las llamadas a las funciones previamente definidas.

En este nivel tenemos que diferenciar entre envío de segmentos **TCP**, y segmentos **UDP**. Esto ocurre ya que el Kernel de Linux utiliza distintas funciones para construir los paquetes IP de estos protocolos. En el caso de **TCP**, se utiliza la función *ip_queue_xmit()* y en el caso de **UDP** se utiliza la función *ip_push_pending_frames()* [1]. Ambas funciones se encuentran en el fichero *net/ipv4/ip_output.c*.

En esas funciones, se ha añadido el código del listado 3.5.

```
1 if (sysctl_ip_netgui) {
2         if (opt)
3                 fill_netgui_opt(opt);
4         else
5                 opt=create_netgui_opt();
6 }
```

Listado 3.5: Llamada a las funciones que añaden la opción de NetGUI.

Este código se encarga de comprobar si el paquete IP que se está procesando contiene alguna opción previamente definida. En caso afirmativo, no es necesario reservar memoria para rellenar los datos puesto que previamente ya se habrá realizado esta operación, pero si no existen opciones en el paquete IP, hay que reservar memoria primero y después rellenar con los datos. De esto se encarga *create_netgui_opt()*.

La función *ip_queue_xmit()* detectará que el paquete que se va a enviar contiene opciones, y por ello las procesará con la función *ip_options_build()* que se encuentra en el fichero *net/ipv4/ip_options.c*. Véase el listado 3.6.

```
1 if (opt && opt->optlen) {
2         iph->ihl += opt->optlen >> 2;
3         ip_options_build(skb, opt, inet->daddr, rt, 0);
4 }
```

Listado 3.6: Comprobación de la existencia de opciones en la función ip_queue_xmit().

Ahora necesitamos saber qué paquetes envían la opción de NetGUI. Para poder detectar cuándo se van a enviar paquetes con la opción de NetGUI hemos añadido el siguiente código a la función *ip_options_build()*. Véase listado 3.7.

```
optptr = opt->is_data ? opt->__data :
         (unsigned char *)&(ip_hdr(skb)[1]);

iph = optptr - sizeof(struct iphdr);

for (l = opt->optlen; l > 0;) {

        optlen = optptr[1];

        if (optlen < 2)
                break;

        if (*optptr == IPOPT_NETGUI) {
                parse_netgui_info(iph,optptr,"SEND",uh,th);
        }

        l -= optlen;
        optptr += optlen;
}
```

Listado 3.7: Comprueba que sea la opción de NetGUI y parsea los datos.

Como se puede observar, cuando se detecta que existe una opción de NetGUI, se invoca a la función *parse_netgui_info()* que se encargará de generar la línea de log correspondiente en el fichero */var/log/messages*.

3.2.2. Modificar el tamaño de la cabecera IP

Las primeras pruebas con la nueva opción de NetGUI se hicieron enviando pequeños paquetes entre distintas máquinas. En principio funcionaba bien, y con *analizadores de tráfico*[2], como TCPDump, se pudo analizar el tráfico y comprobar que en el contenido de la cabecera de IP se encontraba la opción de NetGUI.

Sin embargo, cuando se hicieron pruebas con ficheros en los que se necesitaba enviar

[2]Los *analizadores de tráfico* (sniffers) son programas para la captura de las tramas de red presentes en un determinado interfaz de red de una máquina.

paquetes IP que ocuparan la MTU[3] (Maximum Transfer Unit) de la red, tuvimos problemas. Comprobamos que se perdía parte del contenido de los paquetes, y que no se llegaban a producir transferencias de ficheros satisfactorias. El problema residía en que anteriormente no se tuvo en cuenta que era necesario indicar el nuevo tamaño de la cabecera IP, para que lo tuvieran en cuenta los protocolos de nivel superior, en nuestro caso, **TCP** y **UDP**.

Por lo tanto, se estaban generando unos tramas Ethernet que superaban el valor de MTU de Ethernet en 12 bytes (tamaño de las opciones de NetGUI), y como consecuencia se perdían datos.

En la figura 3.6 se muestra un datagrama IP en el que no se han tenido en cuenta el tamaño de la opción de NetGUI. Como consecuencia supera el MTU y por lo tanto está mal formado.

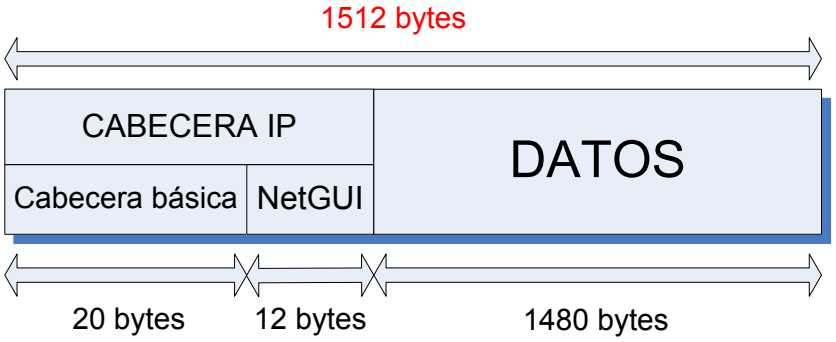


Figura 3.6: *Paquete IP mal formado.*

La solución consistía en modificar el MSS[4] de TCP y UDP.

El tamaño del MSS de TCP debería de ser: MSS = MTU - Tamaño cabecera IP - Tamaño de la opción de NetGUI = 1500 bytes - 20 bytes - 12 bytes = 1468 bytes.

Desde la consola, existe un comando que modifica el MSS de TCP (Listado 3.8).

```
1 iptables -t mangle -A POSTROUTING -p tcp --tcp-flags SYN,RST SYN -o
    eth0 -j TCPMSS --set-mss 1468
```

Listado 3.8: Comando utilizado para modificar el MSS de TCP.

[3]El MTU (Maximun Transfer Unit) es un término que expresa el tamaño en bytes del datagrama más grande que puede pasar por una capa de un protocolo de comunicaciones. En el caso de Ethernet, el MTU máximo es de 1500 bytes.

[4]El MSS (Maximum Segment Size) es el tamaño más grande de datos, especificado en bytes, que un dispositivo de comunicaciones puede manejar en un único trozo, sin fragmentar. Para una comunicación óptima, la suma del número de bytes del segmento de datos y la cabecera debe de ser menor o igual que el MTU.

Esto solucionaba parcialmente el problema, ya que sólo funcionaba para TCP, y además sería necesario ejecutar ese comando cada vez que se arrancara la máquina. Se procedió a buscar la parte de código del Kernel en la que se calcula el MSS de TCP y de UDP, para poder recalcularlo teniendo en cuenta el tamaño de la opción de NetGUI.

La función que se encarga de esto para TCP, se encuentra en el fichero *net/ipv4/tcp_output.c* y tiene por nombre *tcp_mtu_to_mss()*. (Listado 3.9).

```
1  int tcp_mtu_to_mss(struct sock *sk, int pmtu)
2  {
3          struct tcp_sock *tp = tcp_sk(sk);
4          struct inet_connection_sock *icsk = inet_csk(sk);
5          int mss_now;
6
7          /* Calculate base mss without TCP options:
8             It is MMS_S - sizeof(tcphdr) of rfc1122
9           */
10         mss_now = pmtu - icsk->icsk_af_ops->net_header_len - sizeof(
               struct tcphdr);
11
12         /* Clamp it (mss_clamp does not include tcp options) */
13         if (mss_now > tp->rx_opt.mss_clamp)
14                 mss_now = tp->rx_opt.mss_clamp;
15
16         /* Now subtract optional transport overhead */
17         mss_now -= icsk->icsk_ext_hdr_len;
18
19         /************ NetGUI ***********/
20         /* Now subtract NetGUI options */
21         if (sysctl_ip_netgui)
22                 mss_now -= netgui_size;
23         /************ NetGUI ***********/
24
25         /* Then reserve room for full set of TCP options and 8 bytes
               of data */
26         if (mss_now < 48)
27                 mss_now = 48;
28
29         /* Now subtract TCP options size, not including SACKs */
30         mss_now -= tp->tcp_header_len - sizeof(struct tcphdr);
31
32         return mss_now;
33 }
```

Listado 3.9: Función en la que se modifica el MSS de TCP para contemplar a la opción de NetGUI.

En el caso de UDP, se tuvo que modificar la función *ip_append_data()* del fichero *net/ipv4/ip_output.c* que se encarga de generar los paquetes UDP con el tamaño adecuado.

```
1  fragheaderlen = sizeof(struct iphdr) + (opt ? opt->optlen : 0);
2
3  /************ NetGUI ***********/
4
5  if (sysctl_ip_netgui)
6         fragheaderlen += netgui_size; // Para UDP
7
8  /************ NetGUI ***********/
9
10 maxfraglen = ((mtu - fragheaderlen) & ~7) + fragheaderlen;
```

Listado 3.10: Parte del código en la que se calcula el MSS de UDP y se contempla la opción de NetGUI para dicho cálculo.

Con este código ya se consigue que tanto TCP, como UDP contemplen que IP cuenta con la opción de NetGUI en la cabecera, y por lo tanto se forman los paquetes IP de tamaño adecuado.

En la figura 3.7 se muestra un paquete IP bien formado, el cual ocupa como máximo 1500 bytes, teniendo en cuenta la cabecera de IP y la opción de NetGUI.

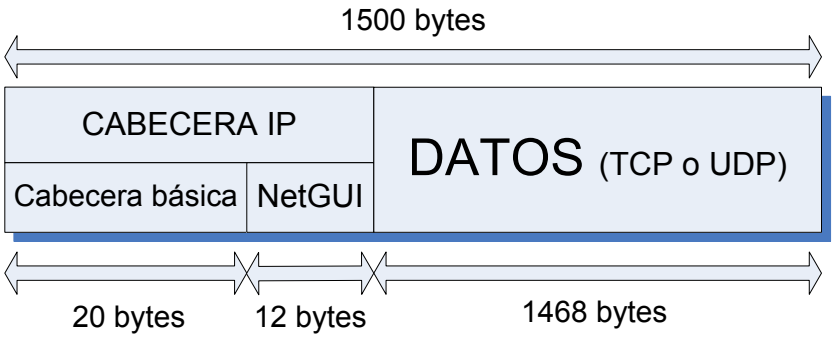


Figura 3.7: *Paquete IP bien formado.*

3.3. Recibir los datos de NetGUI

Una vez que ya tenemos montada la estructura para enviar paquetes IP con el campo de opciones relleno con la información de NetGUI, la siguiente operación es tratar esa información en los nodos que reciben esos paquetes.

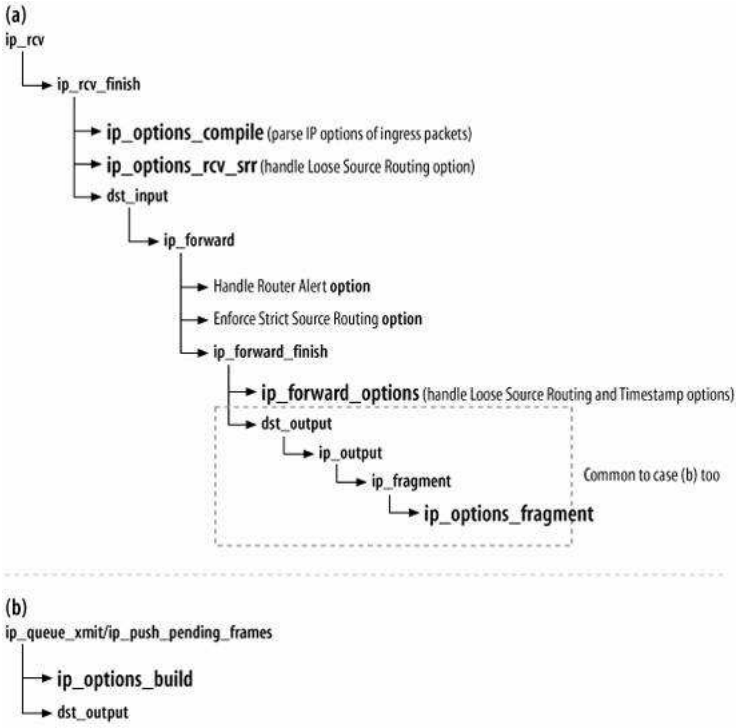


Figura 3.8: *Funciones relacionadas con las opciones IP.*

El la figura 3.8 se puede observar un pequeño esquema con las funciones más relevantes que se utilizan en el kernel de GNU/Linux para procesar las opciones de los datagramas IP. La parte (a) de la figura, representa las funciones que recorren los paquetes que recibe la máquina. La parte (b) de la figura representa las fuciones que se siguen al generar un paquete localmente en la máquina. Hay que diferenciar que *ip_queue_xmit()* es la función utilizada por TCP, mientras que *ip_push_pending_frames()* es la función utilizada por UDP para generar datagramas. Como se puede apreciar en la figura, cuando llega un paquete IP a la función *ip_rcv_finish()* del fichero *ip_input.c* se comprueba que tiene opciones, y después se parsean con la función *ip_options_compile()*.

La función *ip_options_compile()* se puede invocar en dos contextos distintos:

- Desde *ip_rcv_finish* para parsear y validar las opciones de IP de los paquetes que entran al sistema.
- Desde *ip_options_get*, por ejemplo, para parsear los paquetes que han sido definidos a nivel de aplicación con *setsockopt()* y comprobar que son válidos.

En nuestro caso, utilizaremos el primer contexto de la función, para detectar el momento en el que llega un paquete con una opción de NetGUI, y poder procesar los datos recibidos.

En el listado 3.11 se muestra el código de la función *ip_options_compile()* en el que se detecta la opción de NetGUI. Véase el listado .

```
1  case IPOPT_NETGUI:
2            if (!skb && !capable(CAP_NET_RAW)) {
3                      pp_ptr = optptr;
4                      goto error;
5            }
6
7            /* "opt->is_data" distingue entre un paquete recibido y
8               un paquete generado localmente. */
9            if (!opt->is_data)
10                   parse_netgui_info(iph,optptr,"RECV",uh,th);
11                   /* Paquete recibido de otro nodo */
12           break;
```

Listado 3.11: Identificación de la opción de NetGUI en la función ip_options_compile().

Una vez detectado un paquete IP con la opción de NetGUI, hay que tratar el **Identificador de Lamport** para mantener sincronizado el sistema. Por tanto, se comprueba que el identificador de Lamport del nodo que ha recibido el paquete es superior al recibido. En caso contrario, se establece un identificador superior al recibido.

La siguiente tarea consiste en generar la línea de log en el fichero */var/log/messages*. Como se observa en el código fuente anterior, se realiza una llamada a la función *parse_netgui_info()* que será la encargada de comprobar el identificador de Lamport y generar la línea de log.

3.4. Reenviar los datos de NetGUI

En una red en la que haya varios nodos, es muy probable que al establecer una comunicación entre dos máquinas los paquetes tengan que pasar por máquinas intermedias que actúan de *routers*, reenviando los paquetes que no están dirigidos a ellas.

Esto se realiza en el *nivel de red*, donde se realiza la función de encaminamiento basada en la dirección IP destino que lleva el paquete IP. En este nivel, también podemos acceder a las opciones de NetGUI, de modo que nuestra tarea consistirá en establecer un nuevo valor para el identificador de Lamport, y generar una nueva línea en el fichero de log en */var/log/messages*.

La función que se encarga del reenvío de paquetes IP es *ip_forward_finish()* del fichero *net/ipv4/ip_forward.c*. Como se puede observar en el listado 3.12, se hace una llamada a la función *ip_forward_options()* que se encuentra en el fichero *net/ipv4/ip_options.c*.

```
1  static int ip_forward_finish(struct sk_buff *skb)
2  {
3          struct ip_options * opt = &(IPCB(skb)->opt);
4
5          IP_INC_STATS_BH(IPSTATS_MIB_OUTFORWDATAGRAMS);
6
7          if (unlikely(opt->optlen))
8                  ip_forward_options(skb);
9
10         return dst_output(skb);
11 }
```

Listado 3.12: Función ip_forward_finish() que reenvía paquetes IP.

En la función *ip_forward_options()* podemos procesar las opciones de IP. Primero comprobaremos que se encuentra la opción de NetGUI. En caso afirmativo, tenemos que añadir el identificador de Lamport, incrementarlo en el sistema para la próxima vez que se use, y finalmente hacer la llamada a la función *parse_netgui_info()*. Véase listado 3.13.

```
for (l = opt->optlen; l > 0;)  {

        optlen = optptr[1];

        if (optlen < 2)
                break;

        if (*optptr == IPOPT_NETGUI){

                atomic_inc(&netgui_lamport_id); //Incremento el Id
                    de Lamport
                lamport_id = htonl(lamport_id);

                aux = optptr; // Apunto al inicio del campo de las
                    opciones
                aux+=2; // Apunto al identificador de Lamport
                memcpy(aux,&lamport_id,sizeof(lamport_id));

                opt->is_changed = 1; // Para recalcular el checksum
                    de la cabecera
                parse_netgui_info(iph,optptr,"FRWD",uh,th);
        }

        l -= optlen;
        optptr += optlen;
}
```

Listado 3.13: Comprueba que sea la opción de NetGUI y parsea los datos.

Hay que tener en cuenta que al modificar un valor de la cabecera de IP, también hay que recalcular el campo de checksum de la cabecera de IP. Para ello, se utiliza la línea *opt->is_changed = 1;* que más adelante se utilizará para recalcular el checksum.

La función *parse_netgui_info()* simplemente generará la nueva línea de log a partir de los datos del paquete que va a reenviar.

3.5. Activar y desactivar la opción de NetGUI

A nivel de usuario es interesante poder decidir cuándo debe de ser activada o desactivada la modificación de las opciones IP para incluir la opción de NetGUI. Es decir, poder decidir en qué momento los paquetes deben de llevar la información

relacionada con NetGUI. Además, más adelante se verá que esta posibilidad nos ayudará a poder comprobar el funcionamiento de nuestra nueva versión del Kernel.

3.5.1. Variables del Kernel vía */proc*

Dentro del directorio */proc* encontramos dos tipos básicos de información. En primer lugar, para cada proceso activo existe un directorio. Dentro del directorio de cada proceso hay diversos archivos con información específica del proceso.

Adicionalmente, existen una serie de directorios con información acerca de los diferentes módulos del sistema operativo. No todos los parámetros existentes en */proc* son modificables directamente por el usuario. De hecho, la mayoría son valores de sólo lectura.

El usuario **root** del sistema tiene el privilegio de modificar el valor de las variables con permiso de escritura. En el listado 3.14 se observa un ejemplo de cómo se activa en el Kernel ese fichero del sistema, en este caso para ignorar cualquier solicitud de ICMP de eco.

```
1 $ echo 1 > /proc/sys/ipv4/icmp_echo_ignore_all
2 $ cat /proc/sys/ipv4/icmp_echo_ignore_all
3 1
```

Listado 3.14: Modificación de una variable de */proc*

Con esta funcionalidad se pueden modificar diversas variables que utiliza el Kernel, entre ellas, por ejemplo, el **TTL**.

3.5.2. Definición de la variable *sysctl_ip_netgui*

En nuestro caso utilizaremos una variable nueva para poder activar y desactivar la inclusión de las opciones de NetGUI en los paquetes IP. Como se trata de una variable que tiene relación con el protocolo de red la añadiremos en el directorio *net/ipv4*. Para ello previamente hay que definirla en el fichero *net/ipv4/sysctl_net_ipv4.c*. En el listado 3.15 se observa el código necesario para la definición de la variable *sysctl_ip_netgui*.

```
.ctl_name       = NET_IPV4_IP_NETGUI,
.procname       = "ip_netgui",
.data           = &sysctl_ip_netgui,
.maxlen         = sizeof(int),
.mode           = 0644,
.proc_handler   = &proc_dointvec,
```

Listado 3.15: Definición de la variable *sysctl_ip_netgui*

Ahora se puede leer y modificar esta variable desde el código fuente del Kernel, y además también desde la línea de comandos.

Para activar la opción de NetGUI desde la línea de comandos, utilizaremos:

```
$ echo 1 > /proc/sys/net/ipv4/ip_netgui
```

y para desactivarla:

```
$ echo 0 > /proc/sys/net/ipv4/ip_netgui
```

3.6. Interfaz Gráfica

Debido a que este proyecto se centra enormemente en la modificación concreta de una parte del sistema operativo, es difícil poder observar los resultados que ofrece. Por ello, se procedió a implementar una interfaz gráfica (GUI[5]) en la que se puedan representar los resultados. Además también se ha diseñado un algoritmo para poder ordenar los datos recibidos en el **Monitor de NetGUI** con la información de todas las operaciones de envío, reenvío y recepción de cada una de las máquinas de un escenario.

Por defecto, los datos que recibe el nodo **Monitor de NetGUI** se guardan en

[5]Graphical User Interface.

/tmp/netgui. El procedimiento propuesto para el análisis de estos datos es copiar este fichero desde la máquina virtual hasta la máquina anfitriona (Listado 3.16) y posteriormente utilizar la interfaz gráfica en la máquina anfitriona para leer el fichero.

```
1 $ cp /tmp/netgui /hosthome/netgui
```

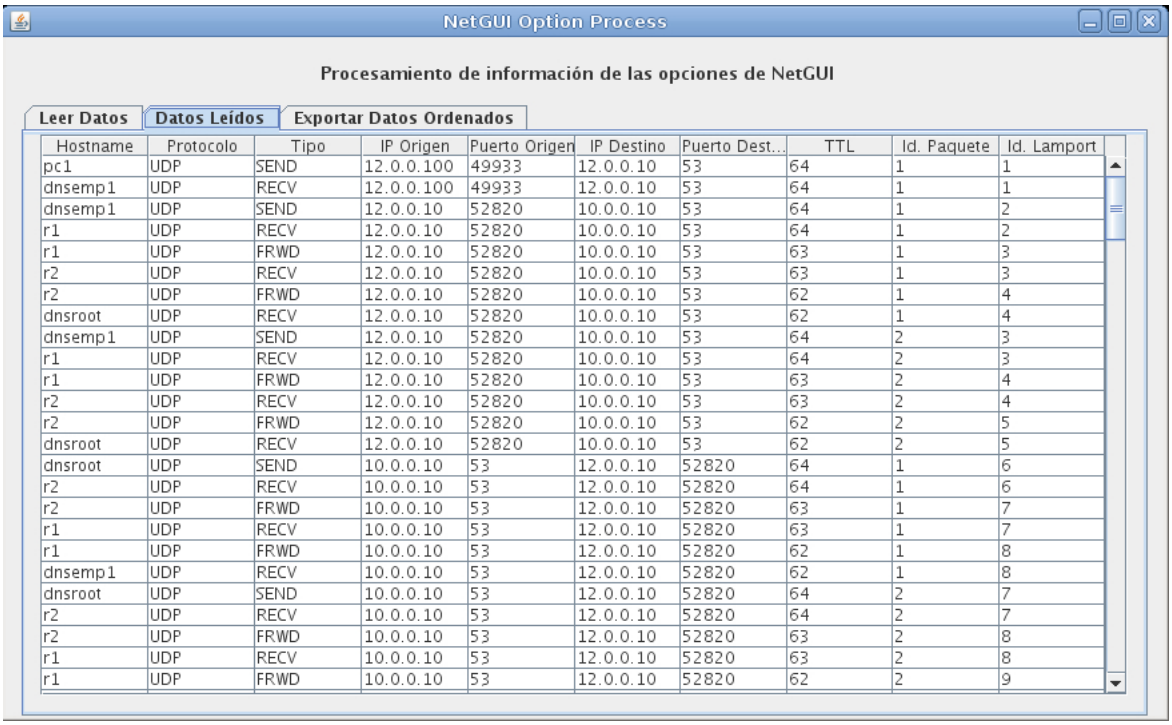
Listado 3.16: Copiar fichero de datos de NetGUI a la máquina anfitriona.

La interfaz gráfica se presenta bastante sencilla. Se compone de tres pestañas distintas (*"Leer Datos"*, *"Datos Leídos"* y *"Exportar Datos Ordenados"*. La pestaña *"Leer Datos"* (Figura 3.9) ofrece la funcionalidad para poder leer un fichero generado en el **Monitor de NetGUI**. Además presenta un cuadro de texto en la que se va presentando un log de los eventos que se suceden en la interfaz.

Figura 3.9: *Pestaña de la Interfaz Gráfica para la lectura del fichero de datos.*

Una vez que se hayan leído satisfactoriamente los datos del fichero, se puede proceder a ver los datos en la pestaña *"Datos Leídos"* (Figura 3.10). En ella se presenta una tabla con los distintos campos (*Hostname, Protocolo, Tipo, IP Origen, Puerto*

Origen, IP Destino, Puerto Destino, TTL, Id. Paquete, Id. Lamport) y los datos se encuentran ordenados por Id. del Paquete y por Id. de Lamport.



Hostname	Protocolo	Tipo	IP Origen	Puerto Origen	IP Destino	Puerto Dest...	TTL	Id. Paquete	Id. Lamport
pc1	UDP	SEND	12.0.0.100	49933	12.0.0.10	53	64	1	1
dnsemp1	UDP	RECV	12.0.0.100	49933	12.0.0.10	53	64	1	1
dnsemp1	UDP	SEND	12.0.0.10	52820	10.0.0.10	53	64	1	2
r1	UDP	RECV	12.0.0.10	52820	10.0.0.10	53	64	1	2
r1	UDP	FRWD	12.0.0.10	52820	10.0.0.10	53	63	1	3
r2	UDP	RECV	12.0.0.10	52820	10.0.0.10	53	63	1	3
r2	UDP	FRWD	12.0.0.10	52820	10.0.0.10	53	62	1	4
dnsroot	UDP	RECV	12.0.0.10	52820	10.0.0.10	53	62	1	4
dnsemp1	UDP	SEND	12.0.0.10	52820	10.0.0.10	53	64	2	3
r1	UDP	RECV	12.0.0.10	52820	10.0.0.10	53	64	2	3
r1	UDP	FRWD	12.0.0.10	52820	10.0.0.10	53	63	2	4
r2	UDP	RECV	12.0.0.10	52820	10.0.0.10	53	63	2	4
r2	UDP	FRWD	12.0.0.10	52820	10.0.0.10	53	62	2	5
dnsroot	UDP	RECV	12.0.0.10	52820	10.0.0.10	53	62	2	5
dnsroot	UDP	SEND	10.0.0.10	53	12.0.0.10	52820	64	1	6
r2	UDP	RECV	10.0.0.10	53	12.0.0.10	52820	64	1	6
r2	UDP	FRWD	10.0.0.10	53	12.0.0.10	52820	63	1	7
r1	UDP	RECV	10.0.0.10	53	12.0.0.10	52820	63	1	7
r1	UDP	FRWD	10.0.0.10	53	12.0.0.10	52820	62	1	8
dnsemp1	UDP	RECV	10.0.0.10	53	12.0.0.10	52820	62	1	8
dnsroot	UDP	SEND	10.0.0.10	53	12.0.0.10	52820	64	2	7
r2	UDP	RECV	10.0.0.10	53	12.0.0.10	52820	64	2	7
r2	UDP	FRWD	10.0.0.10	53	12.0.0.10	52820	63	2	8
r1	UDP	RECV	10.0.0.10	53	12.0.0.10	52820	63	2	8
r1	UDP	FRWD	10.0.0.10	53	12.0.0.10	52820	62	2	9

Figura 3.10: *Pestaña de la Interfaz Gráfica para la presentación de los datos leídos.*

Por último, se encuentra una pestaña que ofrece la funcionalidad de exportar los datos procesados y ordenados de nuevo a un fichero (véase figura 3.11). Esto nos puede servir de ayuda para poder manejar y compartir el fichero. Se compone de un *campo de texto* en el que se debe de introducir la ruta de la máquina en la que se desea guardar el nuevo fichero y un botón para proceder a la exportación.

La interfaz está desarrollada en Java[6] por lo que podemos ejecutarlo en cualquier plataforma o sistema operativo que soporte la máquina virtual de Java.

[6]**Java** es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

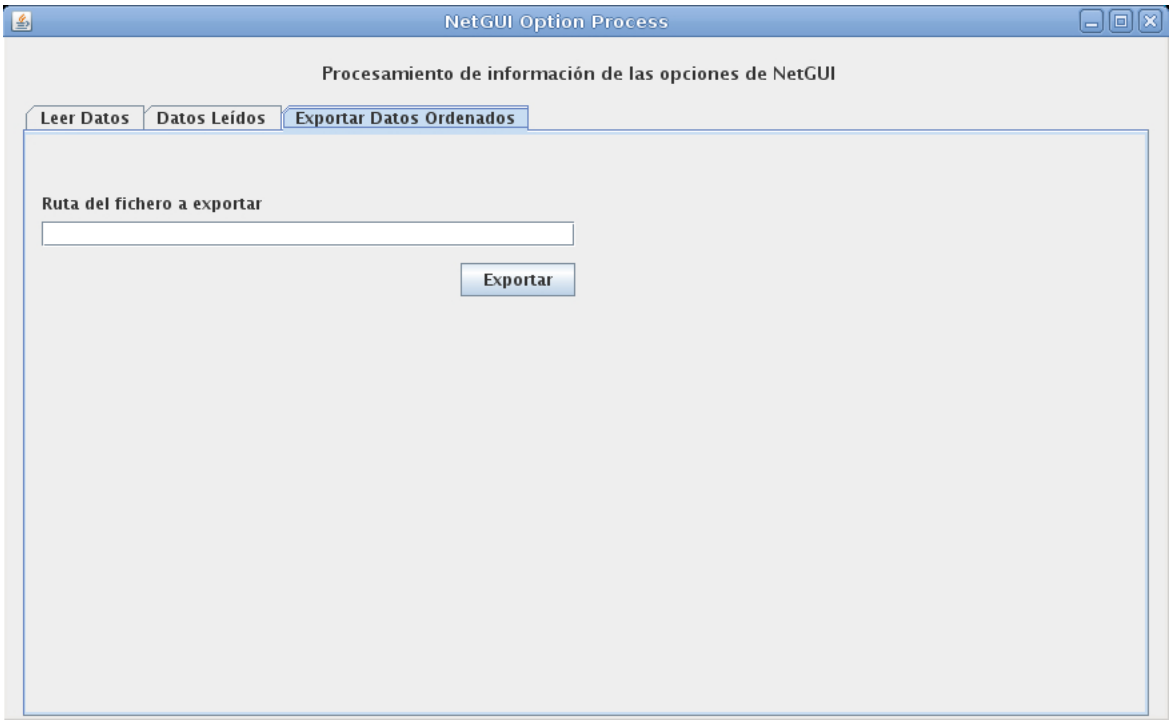


Figura 3.11: *Pestaña de la Interfaz Gráfica para exportar los datos ordenados a un fichero de texto.*

Capítulo 4

Validación

En este capítulo se explicarán en detalle las pruebas realizadas para comprobar el correcto funcionamiento de nuestra implementación. Esta fase consta de tres casos de pruebas, en los que se intenta representar situaciones reales que se pueden encontrar en un escenario de red, y así observar el correcto comportamiento de la implementación realizada tanto en el kernel, como en el Monitor de NetGUI y en la aplicación de ordenación de paquetes IP.

4.1. Primer caso de prueba: Resolución de DNS

Este apartado intenta reproducir una pequeña red en la que los nodos se intercambien mensajes para descargar una página web. Se mostrarán mensajes **UDP** debidos a una operación de resolución del nombre de la página que contiene la página web y mensajes **TCP** debidos a la transferencia **HTTP** de la página web solicitada.

la que se tenga que realizar una resolución de nombres DNS[1], por lo que se utilizará el protocolo **UDP** para posteriormente realizar la descarga de la página, en la que se usará el protocolo **TCP**.

[1]El Domain Name System (DNS) es una base de datos distribuida y jerárquica que almacena información asociada a nombres de dominio en redes como Internet. Aunque como base de datos el DNS es capaz de asociar diferentes tipos de información a cada nombre, los usos más comunes son la asignación de nombres de dominio a direcciones IP y la localización de los servidores de correo electrónico de cada dominio.

4.1.1. Descripción

El propósito principal de esta prueba es que habiendo montado un sistema con varios nodos conectados en red, se comuniquen entre sí con el fin de que se pueda observar el intercambio de mensajes utilizando diferentes protocolos de nivel de aplicación y transporte entre las distintas máquinas. Para ello se han montado cinco servidores de DNS (*dnsroot, dnscom, dnsnet, dnsemp1 y dnsemp2*), un servidor web Apache en uno de los nodos (*pc2*) y un nodo (*pc1*) que actuará como cliente y solicitará la página web que está alojada en pc2. Además la red contará con 4 routers (*r1, r2, r3 y r4*) y la máquina **Monitor de NetGUI** que se encargará de recibir la información del tráfico que haya en la red. (Var figura 4.1).

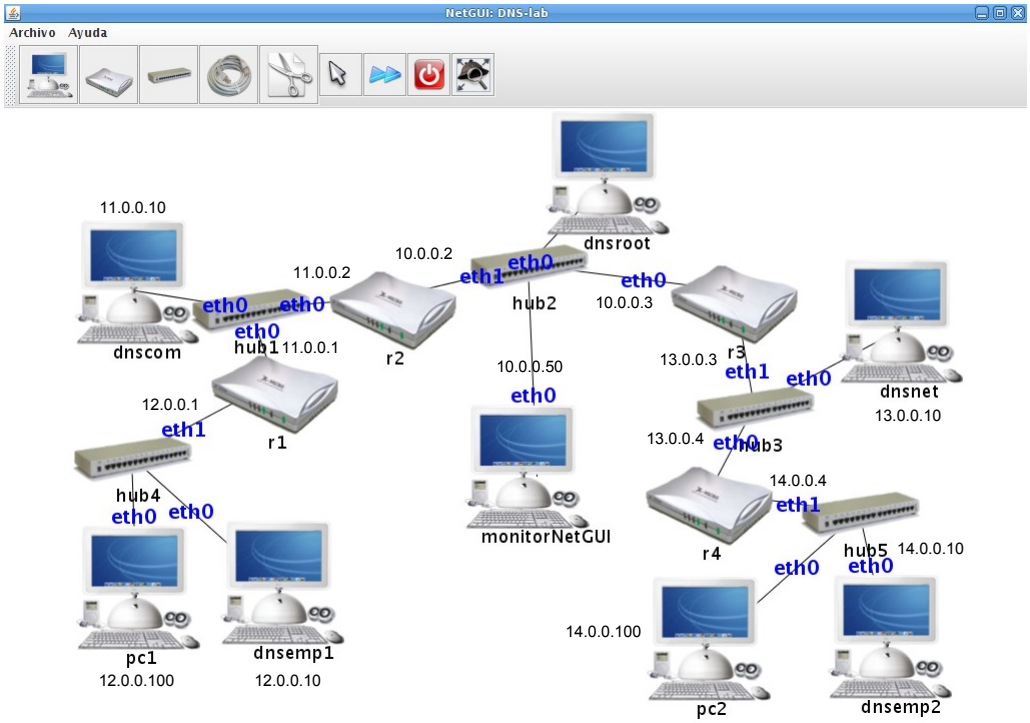


Figura 4.1: *Escenario de la primero prueba.*

La idea es que al intentar descargar el contenido de la página web alojada en *pc2* desde *pc1* se pueda observar la resolución de nombres mediante mensajes UDP, y finalmente la descarga del contenido de la página mediante mensajes TCP. Al haber varios servidores de DNS, si una máquina hace una solicitud de resolución de un nombre a uno de ellos, y éste no sabe cómo resolver el nombre, este servidor de DNS hará las solicitudes necesarios al resto de los servidores de DNS hasta que encuentre

la IP que busca, para posteriormente responder a la máquina que hizo la solicitud inicial.

Los mensajes a nivel de transporte que se deberían de mostrar al solicitar la descarga de la página web son los siguientes:

1. Petición **UDP** desde *pc1* a *dnsemp1* para resolver *pc2.emp2.net*.
2. Petición **UDP** desde *dnsemp1* a *dnsroot*.
3. Respuesta **UDP** desde *dnsroot* a *dnsemp1*.
4. Petición **UDP** desde *dnsemp1* a *dnsnet*.
5. Respuesta **UDP** desde *dnsnet* a *dnsemp1*.
6. Petición **UDP** desde *dnsemp1* a *dnsemp2*.
7. Respuesta **UDP** desde *dnsemp2* a *dnsemp1*.
8. Petición **UDP** desde *dnsemp1* a *pc1*.
9. Petición **TCP** desde *pc1* a *pc2* para solicitar la página web vía HTTP.
10. Respuesta **TCP** desde *pc2* a *pc1* con la página web solicitada vía HTTP.

4.1.2. Resultados

A continuación se observan algunos de los mensajes capturados por el *Monitor de NetGUI* en la prueba. En esta primera traza se aprecia la solicitud de DNS que solicita *pc1* a *dnsemp1*.

```
Jun 15 15:23:45 pc1 kernel: NETGUI, UDP, SEND, 1, 1, 12.0.0.100:49933, 12.0.0.10:53, 64
Jun 15 15:23:45 dnsemp1 kernel: NETGUI, UDP, RECV, 1, 1, 12.0.0.100:49933, 12.0.0.10:53, 64
Jun 15 15:23:45 dnsemp1 kernel: NETGUI, UDP, SEND, 2, 1, 12.0.0.10:52820, 10.0.0.10:53, 64
Jun 15 15:23:45 r1 kernel: NETGUI, UDP, RECV, 2, 1, 12.0.0.10:52820, 10.0.0.10:53, 64
Jun 15 15:23:45 r1 kernel: NETGUI, UDP, FRWD, 3, 1, 12.0.0.10:52820, 10.0.0.10:53, 63
Jun 15 15:23:45 r2 kernel: NETGUI, UDP, RECV, 3, 1, 12.0.0.10:52820, 10.0.0.10:53, 63
Jun 15 15:23:45 r2 kernel: NETGUI, UDP, FRWD, 4, 1, 12.0.0.10:52820, 10.0.0.10:53, 62
Jun 15 15:23:45 dnsroot kernel: NETGUI, UDP, RECV, 4, 1, 12.0.0.10:52820, 10.0.0.10:53, 62
```

En esta otra traza, se observa cómo *pc1* realiza las peticiones al puerto 80 de *pc2*, y éste le contesta.

Jun 15 15:23:45 pc1 kernel: NETGUI, TCP, SEND, 45, 6, 12.0.0.100:45005, 14.0.0.100:80, 64
Jun 15 15:23:45 r1 kernel: NETGUI, TCP, RECV, 45, 6, 12.0.0.100:45005, 14.0.0.100:80, 64
Jun 15 15:23:45 r1 kernel: NETGUI, TCP, FRWD, 46, 6, 12.0.0.100:45005, 14.0.0.100:80, 63
Jun 15 15:23:45 r2 kernel: NETGUI, TCP, RECV, 46, 6, 12.0.0.100:45005, 14.0.0.100:80, 63
Jun 15 15:23:45 r2 kernel: NETGUI, TCP, FRWD, 47, 6, 12.0.0.100:45005, 14.0.0.100:80, 62
Jun 15 15:23:45 r3 kernel: NETGUI, TCP, RECV, 47, 6, 12.0.0.100:45005, 14.0.0.100:80, 62
Jun 15 15:23:45 r3 kernel: NETGUI, TCP, FRWD, 48, 6, 12.0.0.100:45005, 14.0.0.100:80, 61
Jun 15 15:23:45 r4 kernel: NETGUI, TCP, RECV, 48, 6, 12.0.0.100:45005, 14.0.0.100:80, 61
Jun 15 15:23:45 r4 kernel: NETGUI, TCP, FRWD, 49, 6, 12.0.0.100:45005, 14.0.0.100:80, 60
Jun 15 15:23:45 pc2 kernel: NETGUI, TCP, RECV, 49, 6, 12.0.0.100:45005, 14.0.0.100:80, 60
Jun 15 15:23:45 pc2 kernel: NETGUI, TCP, SEND, 50, 1, 14.0.0.100:80, 12.0.0.100:45005, 64
Jun 15 15:23:45 r4 kernel: NETGUI, TCP, RECV, 50, 1, 14.0.0.100:80, 12.0.0.100:45005, 64
Jun 15 15:23:45 r4 kernel: NETGUI, TCP, FRWD, 51, 1, 14.0.0.100:80, 12.0.0.100:45005, 63
Jun 15 15:23:45 r3 kernel: NETGUI, TCP, RECV, 51, 1, 14.0.0.100:80, 12.0.0.100:45005, 63
Jun 15 15:23:45 r3 kernel: NETGUI, TCP, FRWD, 52, 1, 14.0.0.100:80, 12.0.0.100:45005, 62
Jun 15 15:23:45 r2 kernel: NETGUI, TCP, RECV, 52, 1, 14.0.0.100:80, 12.0.0.100:45005, 62
Jun 15 15:23:45 r2 kernel: NETGUI, TCP, FRWD, 53, 1, 14.0.0.100:80, 12.0.0.100:45005, 61
Jun 15 15:23:45 r1 kernel: NETGUI, TCP, RECV, 53, 1, 14.0.0.100:80, 12.0.0.100:45005, 61
Jun 15 15:23:45 r1 kernel: NETGUI, TCP, FRWD, 54, 1, 14.0.0.100:80, 12.0.0.100:45005, 60
Jun 15 15:23:45 pc1 kernel: NETGUI, TCP, RECV, 54, 1, 14.0.0.100:80, 12.0.0.100:45005, 60

Como se ha podido apreciar en esta fase se diferencia claramente cada uno de los mensajes que se han transmitido, y su orden con respecto al resto de los datagramas capturados en la red. Por todo esto se puede concluir que esta prueba ha resultado satisfactoria.

4.2. Segundo caso de prueba: Cambio de ruta

Esta vez se ha elegido un escenario más sencillo que el anterior. La intención es la de modificar la ruta por la que viajan los datagramas IP. Para ello, se ha diseñado un escenario de red en el que existen dos posibles caminos para comunicar pc1 y pc2.

4.2.1. Descripción

Se han elegido 2 nodos que serán los que se intercambien datagramas. La máquina *pc2* actuará como servidor enviando un fichero de texto a la máquina *pc1* que actuando de cliente recibirá el fichero. Ambas máquinas se pueden comunicar utilizando dos caminos diferentes, a través de *r1* o a través de *r2*. (Ver figura 4.2). Inicialmente se ha establecido al nodo *r1* como la máquina por la que *pc1* enviará sus datagramas IP a *pc2*. Por el contrario, *pc2* utilizará a *r2* para transmitir sus mensajes IP a *pc1*.

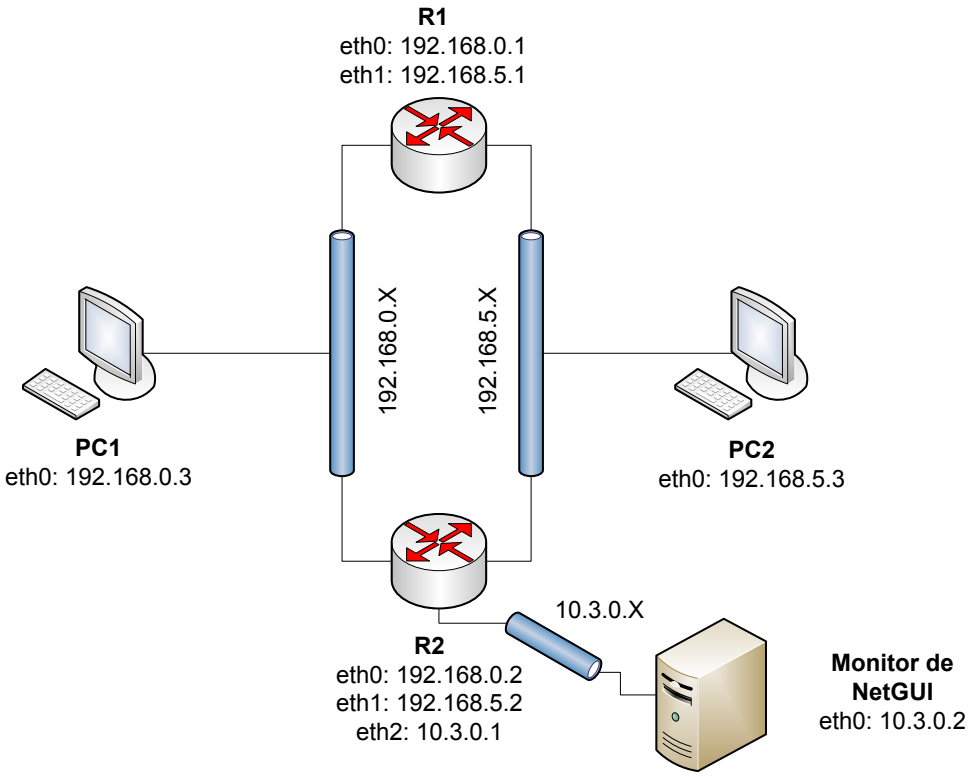


Figura 4.2: *Escenario de la segunda prueba.*

Se iniciará a continuación una transmisión de un fichero entre *pc2* y *pc1*, y en mitad del proceso se cambiará la ruta que tiene asignada la máquina *pc2* para que envíe a través de *r1*.

Para realizar esta prueba, se ha utilizado el programa *netcat* que permite abrir puerto TCP/UDP y además existe un parámetro con el que se puede definir el tiempo que debe de pasar entre cada uno de los datagramas enviados. Esto nos da la flexibilidad suficiente para modificar la ruta en mitad de la transmisión.

En el listado 4.1 se observa el comando ejecutado por *pc1* para abrir el puerto 5555 de TCP donde recibirá el fichero. En el listado 4.2 se observa el comando ejecutado por *pc2* para enviar un fichero a *pc1*.

```
1 $ netcat -l -p 5555
```

Listado 4.1: Comando ejecutado en pc1 para recibir el fichero.

```
1 $ cat fichero | netcat -i 1 192.168.0.3 5555
```

Listado 4.2: Comando ejecutado en pc2 para enviar el fichero.

En cualquier caso, la máquina *Monitor de NetGUI* está presente en el escenario, y como de costumbre recibirá la información del tráfico de la red.

4.2.2. Resultados

A continuación se observan algunos de los mensajes capturados por el *Monitor de NetGUI* en la prueba. En la traza que se muestra a continuación se observa que en el primer envío *pc2* utiliza al router *r2* para alcanzar el destino (*pc1*). En el siguiente envío que realiza *pc2* se ha cambiado la ruta, y se observa que utiliza al router *r1* para que el datagrama terminara en *pc1*.

Jun 13 17:44:31 pc2 kernel: NETGUI, TCP, SEND, 129, 32, 192.168.5.3:39609, 192.168.0.3:5555, 64
Jun 13 17:44:31 r2 kernel: NETGUI, TCP, RECV, 129, 32, 192.168.5.3:39609, 192.168.0.3:5555, 64
Jun 13 17:44:31 r2 kernel: NETGUI, TCP, FRWD, 130, 32, 192.168.5.3:39609, 192.168.0.3:5555, 63
Jun 13 17:44:31 pc1 kernel: NETGUI, TCP, RECV, 130, 32, 192.168.5.3:39609, 192.168.0.3:5555, 63
Jun 13 17:44:31 pc1 kernel: NETGUI, TCP, SEND, 131, 35, 192.168.0.3:5555, 192.168.5.3:39609, 64
Jun 13 17:44:31 r1 kernel: NETGUI, TCP, RECV, 131, 35, 192.168.0.3:5555, 192.168.5.3:39609, 64
Jun 13 17:44:31 r1 kernel: NETGUI, TCP, FRWD, 132, 35, 192.168.0.3:5555, 192.168.5.3:39609, 63
Jun 13 17:44:31 pc2 kernel: NETGUI, TCP, RECV, 132, 35, 192.168.0.3:5555, 192.168.5.3:39609, 63
Jun 13 17:44:32 pc2 kernel: NETGUI, TCP, SEND, 133, 33, 192.168.5.3:39609, 192.168.0.3:5555, 64
Jun 13 17:44:32 r1 kernel: NETGUI, TCP, RECV, 133, 33, 192.168.5.3:39609, 192.168.0.3:5555, 64
Jun 13 17:44:32 r1 kernel: NETGUI, TCP, FRWD, 134, 33, 192.168.5.3:39609, 192.168.0.3:5555, 63
Jun 13 17:44:32 pc1 kernel: NETGUI, TCP, RECV, 134, 33, 192.168.5.3:39609, 192.168.0.3:5555, 63

```
Jun 13 17:44:32 pc1 kernel: NETGUI, TCP, SEND, 135, 36, 192.168.0.3:5555, 192.168.5.3:39609, 64
Jun 13 17:44:32 r1 kernel: NETGUI, TCP, RECV, 135, 36, 192.168.0.3:5555, 192.168.5.3:39609, 64
Jun 13 17:44:32 r1 kernel: NETGUI, TCP, FRWD, 136, 36, 192.168.0.3:5555, 192.168.5.3:39609, 63
Jun 13 17:44:32 pc2 kernel: NETGUI, TCP, RECV, 136, 36, 192.168.0.3:5555, 192.168.5.3:39609, 63
```

En esta fase de las pruebas se ha comprobado satisfactoriamente el buen funcionamiento de nuestra aplicación en situaciones en las que se modifica *en vivo* las rutas que tienen por defecto las máquinas, y la facilidad que ofrecen las trazas generadas en el **Monitor de NetGUI** para analizar lo ocurrido en la red.

4.3. Tercer caso de prueba: Envíos simultáneos desde máquinas distintas

Por último se ha optado por realizar las pruebas bajo una topología sencilla en la que como propósito principal se desea comprobar el funcionamiento de nuestro proyecto fin de carrera en una situación en la que haya dos nodos enviando datagramas IP simultáneamente a un mismo nodo. Para realizar este caso de prueba se han utilizado 3 máquinas (*pc1, pc2 y pc3*), un router (*r1*) y el nodo **Monitor de NetGUI**.

4.3.1. Descripción

La idea es que exista un nodo ejecutando una aplicación servidora escuchando en 2 puertos distintos, y en cada uno de ellos se estén recibiendo datos de distintas máquinas. De este modo se podrán observar los datagramas que se intercambien en la red, y se podrán diferenciar los que se envían desde máquinas distintas. La máquina *pc1* actuará como nodo receptor, y las máquinas *pc2 y pc3* serán las encargadas de enviar los mensajes a *pc1*. Cabe destacar que se deberá de poder comprobar que los mensajes que se envían desde *pc2* seguirán una ruta distina a los que se envíen desde *pc3*.

Para realizar esta prueba, también se ha utilizado el programa *netcat* como en el caso de prueba anterior. En *pc1* se han ejecutado dos instancias de este programa, una por cada puerto. En *pc1* se ha ejecutado *netcat* en modo servidor, escuchando en dos puertos diferentes (5555 y 6666). Véase listado 4.3.

```
1 $ netcat -l -p 5555 &
2 $ netcat -l -p 6666 &
```

Listado 4.3: Comandos ejecutados en pc1 para recibir los ficheros.

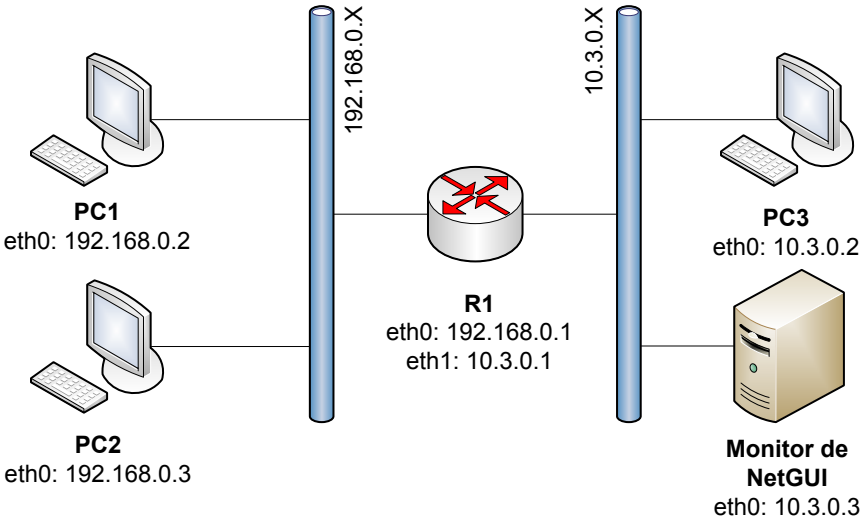


Figura 4.3: *Escenario de la tercera prueba.*

Tanto en *pc2* y *pc3* se ha ha ejecutado netcat en modo cliente enviando datos a *pc1*. Véase listados 4.4 y 4.5

```
1 $ cat fichero | netcat -i 1 192.168.0.2 5555
```

Listado 4.4: Comando ejecutado en pc2 para enviar el fichero.

```
1 $ cat fichero | netcat -i 1 192.168.0.2 6666
```

Listado 4.5: Comando ejecutado en pc3 para enviar el fichero.

Por supuesto, la máquina **Monitor de NetGUI** está presente en la topología de red, y como en los casos anteriores se encargará de recibir toda la información de NetGUI gestionada en el resto de los nodos.

4.3.2. Resultados

A continuación se muestran algunos de los datagramas que se han intercambiado en la red y que han sido recibidos en el *Monitor de NetGUI*. En la traza se puede apreciar que *pc3* está enviando mensajes al puerto *6666* de *pc1*, y que *pc2* envía mensajes al puerto *5555*. Gracias a los puertos podemos diferenciar claramente la procedencia del mensaje. También se puede observar que los mensajes que envía *pc3* pasan por *r1* para llegar al destino, y que sin embargo, como era de esperar, los datagramas desde *pc2* no pasan por ninguna máquina intermedia.

```
Jun 21 23:08:04 pc3 kernel: NETGUI, TCP, SEND, 27, 5, 10.3.0.2:37908, 192.168.0.2:6666, 64
Jun 21 23:08:04 r1 kernel: NETGUI, TCP, RECV, 27, 5, 10.3.0.2:37908, 192.168.0.2:6666, 64
Jun 21 23:08:04 r1 kernel: NETGUI, TCP, FRWD, 28, 5, 10.3.0.2:37908, 192.168.0.2:6666, 63
Jun 21 23:08:04 pc1 kernel: NETGUI, TCP, RECV, 28, 5, 10.3.0.2:37908, 192.168.0.2:6666, 63
Jun 21 23:08:04 pc2 kernel: NETGUI, TCP, SEND, 27, 14, 192.168.0.3:59613, 192.168.0.2:5555, 64
Jun 21 23:08:04 pc1 kernel: NETGUI, TCP, RECV, 27, 14, 192.168.0.3:59613, 192.168.0.2:5555, 64
Jun 21 23:08:04 pc1 kernel: NETGUI, TCP, SEND, 29, 14, 192.168.0.2:6666, 10.3.0.2:37908, 64
Jun 21 23:08:04 r1 kernel: NETGUI, TCP, RECV, 29, 14, 192.168.0.2:6666, 10.3.0.2:37908, 64
Jun 21 23:08:04 r1 kernel: NETGUI, TCP, FRWD, 30, 14, 192.168.0.2:6666, 10.3.0.2:37908, 63
Jun 21 23:08:04 pc3 kernel: NETGUI, TCP, RECV, 30, 14, 192.168.0.2:6666, 10.3.0.2:37908, 63
Jun 21 23:08:04 pc1 kernel: NETGUI, TCP, SEND, 30, 15, 192.168.0.2:5555, 192.168.0.3:59613, 64
Jun 21 23:08:04 pc2 kernel: NETGUI, TCP, RECV, 30, 15, 192.168.0.2:5555, 192.168.0.3:59613, 64
Jun 21 23:08:05 pc3 kernel: NETGUI, TCP, SEND, 31, 6, 10.3.0.2:37908, 192.168.0.2:6666, 64
Jun 21 23:08:05 r1 kernel: NETGUI, TCP, RECV, 31, 6, 10.3.0.2:37908, 192.168.0.2:6666, 64
Jun 21 23:08:05 r1 kernel: NETGUI, TCP, FRWD, 32, 6, 10.3.0.2:37908, 192.168.0.2:6666, 63
Jun 21 23:08:05 pc1 kernel: NETGUI, TCP, RECV, 32, 6, 10.3.0.2:37908, 192.168.0.2:6666, 63
Jun 21 23:08:05 pc2 kernel: NETGUI, TCP, SEND, 31, 15, 192.168.0.3:59613, 192.168.0.2:5555, 64
Jun 21 23:08:05 pc1 kernel: NETGUI, TCP, RECV, 31, 15, 192.168.0.3:59613, 192.168.0.2:5555, 64
Jun 21 23:08:05 pc1 kernel: NETGUI, TCP, SEND, 33, 16, 192.168.0.2:6666, 10.3.0.2:37908, 64
Jun 21 23:08:05 r1 kernel: NETGUI, TCP, RECV, 33, 16, 192.168.0.2:6666, 10.3.0.2:37908, 64
Jun 21 23:08:05 r1 kernel: NETGUI, TCP, FRWD, 34, 16, 192.168.0.2:6666, 10.3.0.2:37908, 63
Jun 21 23:08:05 pc3 kernel: NETGUI, TCP, RECV, 34, 16, 192.168.0.2:6666, 10.3.0.2:37908, 63
Jun 21 23:08:05 pc1 kernel: NETGUI, TCP, SEND, 34, 17, 192.168.0.2:5555, 192.168.0.3:59613, 64
Jun 21 23:08:05 pc2 kernel: NETGUI, TCP, RECV, 34, 17, 192.168.0.2:5555, 192.168.0.3:59613, 64
Jun 21 23:08:06 pc3 kernel: NETGUI, TCP, SEND, 35, 7, 10.3.0.2:37908, 192.168.0.2:6666, 64
Jun 21 23:08:06 r1 kernel: NETGUI, TCP, RECV, 35, 7, 10.3.0.2:37908, 192.168.0.2:6666, 64
Jun 21 23:08:06 r1 kernel: NETGUI, TCP, FRWD, 36, 7, 10.3.0.2:37908, 192.168.0.2:6666, 63
Jun 21 23:08:06 pc1 kernel: NETGUI, TCP, RECV, 36, 7, 10.3.0.2:37908, 192.168.0.2:6666, 63
```

Se comprueba que nuestro proyecto fin de carrera funciona como era de esperar, y que como en el resto de los casos, se distingue cada uno de los datagramas que viajan en la red, el orden, y el camino recorrido por cada uno de ellos.

Capítulo 5

Conclusiones y trabajos futuros

En este capítulo se resumen las conclusiones a las que se ha llegado tras la realización del diseño, implementación y validación de este proyecto. Además se exponen las posibles líneas futuras en las que se puede seguir investigando para que proyectos como éste avancen en la forma de mostrar más claramente el funcionamiento de las redes de ordenadores.

5.1. Conclusiones

Repasando los objetivos marcados en el capítulo 2 y teniendo en cuenta el principal objetivo marcado para este proyecto vamos a repasar punto por punto la medida en la que se han satisfecho dichos objetivos. El objetivo principal de este proyecto era tratar de añadir información dentro de los paquetes IP para posteriormente ser capaces de *reconstruir* el orden y las rutas que han seguido cada uno de los datagramas generados en la red.

El objetivo principal estaba articulado en cuatro etapas principales que se satisficieron con los siguientes subobjetivos:

El primer subobjetivo, *Definición de la información adicional que viaja en los datagramas IP*, se ha descrito en el capítulo 3. Se ha diseñado un esquema para un tipo de opciones de los paquetes IP en el que se añaden dos identificadores que viajan en todos los datagramas IP, y con los que nos es posible averiguar el orden y recorrido que han seguido cada uno de ellos. Para ello se ha utilizado una técnica para la sincronización de sistemas distribuídos que nos ha facilitado la tarea de ordenación de los eventos que ocurren entre máquinas diferentes y que están relacionados con el envío, reenvío y recepción de mensajes.

El segundo subobjetivo, la modificación del Kernel de UML, fue la materialización del diseño propuesto en el primer subobjetivo. Con ello se ha conseguido que cada datagrama que se genere en una máquina lleve incorporadas las opciones de NetGUI de forma nativa, y que además, se pueda decidir en el momento en el que se desea *habilitar/deshabilitar* estas opciones. De este modo el usuario tiene el control absoluto sobre la generación de las opciones de NetGUI.

El tercer subobjetivo, consistía en modificar el kernel de UML para que se almacene los datos individuales de cada paquete procesado en cada uno de los nodos. Esto nos ayuda a poder observar fácilmente en cada una de las máquinas lo que ocurre en la red.

El cuarto subobjetivo, la distribución de los datos y posterior concentración en el **Monitor de NetGUI** es la parte en la que se logra conseguir el objetivo principal del proyecto. Con ello, se consigue la completa monitorización de los mensajes que se han intercambiado en la red.

El capítulo 4 de *Validación* nos ha servido para poder comprobar que lo implementado funciona correctamente y que se puede llevar a la práctica de forma sencilla. Además también nos ha servido para poder encontrar posibles mejoras que se podrían aplicar a este proyecto.

Además se ha implementado una interfaz gráfica que facilita aún más el estudio de los datos generados en las implementaciones anteriores. En esta interfaz se ha implementado un algoritmo de ordenación para que sea más sencillo poder observar los datagramas ordenándolos por *Identificador de Lamport* e *Identificador de Paquete*.

A modo de balance concluimos que la nueva implementación del kernel para la monitorización de paquetes IP es robusta y fiable. Nos aporta información acerca de lo que ocurre en la red que anteriormente era difícil o imposible conseguir. Con ello hemos conseguido mejorar la capacidad de observación que tenemos frente a cualquier escenario de red, y así poder afrontar cualquier estudio relacionado.

A nivel personal este proyecto me ha aportando la experiencia necesaria para poder afrontar posibles proyectos relacionados con implementaciones sobre sistemas operativos, además he fortalecido mis conocimientos de redes y más en concreto sobre la arquitectura de protocolos TCP/IP.

5.2. Trabajo futuro

En este apartado se detallan algunas posibles mejoras que podrían realizarse sobre este proyecto y que pueden servir como nuevas líneas de invetigación para futuros trabajos.

La evolución natural de este proyecto, consiste en incluirlo como parte del sistema que ofrece NetGUI. Desde un principio ha sido concebido para usarse en entornos didácticos con el fín de que los alumnos puedan estudiar desde un aspecto más práctico las redes.

Se pueden incluir mejoras en cuanto a añadir las opciones de IP no sólo en los segmentos TCP y paquetes UDP como lo hecho hasta ahora, sino también en otros paquetes como pueden ser ICMP[1] o IGMP[2]. Con esta propuesta, se podría tener aún más información acerca de lo que ocurre en la red, y se podrían estudiar estos protocolos más en detalle.

También se podría mejorar la interfaz gráfica de NetGUI para representar paso a paso a través de flechas el intercambio de mensajes que se realizan en un escenario de red determinado.

[1]El **Protocolo de Mensajes de Control de Internet** o **ICMP** es el subprotocolo de control y notificación de errores del Protocolo de Internet (IP). Como tal, se usa para enviar mensajes de error, indicando por ejemplo que un servicio determinado no está disponible o que un router o host no puede ser localizado.

[2]El protocolo de red **IGMP** se utiliza para intercambiar información acerca del estado de pertenencia entre enrutadores IP que admiten la multidifusión y miembros de grupos de multidifusión.

Bibliografía

[1] Christian Benvenuti. *Understanding Linux Network Internals*. ISBN 0-596-00255-6. O'Really, 2006.

[2] Santiago Carot Nemesio, Pedro de las Heras Quirós, Eva M. Castro Barbero, and José A. Centeno González. Early experiences with netgui laboratories.

[3] Richard Stevens. *TCP IP Illustrated, Volume 1 (The Protocols)*. ISBN 0-201-63346-9. Addison-Wesley, 1993.

[4] Andrew S. Tanenbaum. *Sistemas Operativos Distribuidos*. ISBN 968-880-627-7. Prentice-Hall Hisp, 1994.