ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

IMPLEMENTACIÓN DE UNA APLICACIÓN PARA NETWORK CODING UTILIZANDO REDES DEFINIDAS POR SOFTWARE

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

VÍCTOR JAVIER PILCO ESCOBAR

vicj_1292@hotmail.com

DIRECTOR: ING. RAUL DAVID MEJÍA NAVARRETE, M.Sc.

david.mejia@epn.edu.ec

Quito, abril 2017

DECLARACIÓN

Yo, Víctor Javier Pilco Escobar, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

VÍCTOR JAVIER PILCO ESCOBAR

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Víctor Javier Pilco Escobar, bajo mi supervisión.

ING. DAVID MEJIA, M.Sc.
DIRECTOR DEL TRABAJO DE TITULACIÓN

AGRADECIMIENTOS

Quiero agradecer a mis padres, Víctor Pilco y Pilar Escobar, cuyo apoyo durante toda mi carrera ha sido de mucha ayuda en los momentos más difíciles. A mi hermano Adrián Pilco que me acompaño y me ayudo a ser más responsable y decidido. A Viviana Moya, por haber sido mi compañera y mejor amiga durante toda mi carrera, y me acompaño en las buenas y las malas siempre dándome ánimos para salir adelante. A Roberto Navarrete por tenerme paciencia y ayudarme a ser más independiente. A toda mi familia y amigos, que con su cariño me daban fuerzas para continuar. A mis profesores cuyos conocimientos me hicieron amar esta carrera y a mi director el Ing. David Mejía por haberme dado tantas muy buenas oportunidades durante mis estudios.

Muchas Gracias.

DEDICATORIA

A mis padres por todo su apoyo, cariño y paciencia durante toda mi vida.

CONTENIDO

RESUMEN

El presente trabajo presenta los resultados de un proyecto de investigación para el cual se implementa una solución de software que utiliza codificación de red como una herramienta para la recuperación de paquetes perdidos en una Red Definida por Software.

Inicialmente se analiza el funcionamiento de las Redes Definidas por Software, el protocolo de comunicaciones OpenFlow, se realiza una introducción a Mininet como herramienta para la simulación de redes de datos, un breve análisis del *framework* Ryu y el funcionamiento básico de la codificación de red, así como también se plantea su utilización como un mecanismo para la recuperación de tramas en una red de manera transparente para los equipos terminales.

A continuación, se presenta la metodología empleada en este proyecto, el diseño de una red con retardos y errores en los enlaces, una aplicación desarrollada para el controlador Ryu y aplicaciones adicionales escritas en el lenguaje de programación Python con el fin de realizar las tareas necesarias para la codificación de red y recuperación de paquetes perdidos.

Finalmente, se presentan los resultados de las pruebas ejecutadas en el escenario implementado, se presenta la cantidad de datos enviados, recibidos, perdidos y recuperados, y en base a estos resultados se establece el porcentaje de recuperación de datos gracias a la aplicación.

PRESENTACIÓN

El presente proyecto se compone de cuatro capítulos:

En el Capítulo 1 se presenta un marco teórico compuesto de: una introducción a las Redes Definidas por Software, sus componentes principales y estructura básica; un resumen del protocolo OpenFlow, sus componentes, etapas de comunicación y mensajes básicos; un resumen del controlador de SDN Ryu y el entorno de simulación Mininet; así como también una explicación detallada de la topología de red tipo *butterfly*; se presenta además una introducción al concepto de codificación de red y se presenta un ejemplo de su aplicación; finalmente, se presenta una explicación de la utilización de la codificación de red orientada a la recuperación de datos en una red tipo *butterfly* con enlaces que poseen errores.

En el Capítulo 2 se presenta el diseño e implementación de los distintos componentes del ambiente desarrollado: las características de las 7 máquinas virtuales utilizadas, la red virtual con topología tipo *butterfly* con Mininet; dos aplicaciones para el controlador Ryu que generan reglas de funcionamiento para transmisión tradicional y transmisión utilizando codificación de red; y dos aplicaciones utilizadas para realizar el procesamiento necesario para codificar tramas y recuperar tramas perdidas. Finalmente, se presentan las pruebas de funcionamiento de todos los componentes del ambiente de red en 3 distintos escenarios de análisis.

En el Capítulo 3 se presentan los resultados de las pruebas realizadas en el ambiente de red implementado para distintas tasas de pérdidas configuradas en los enlaces de la red; se presenta una comparación entre los resultados esperados y los resultados obtenidos, así como también la discusión y análisis de los mismos.

En el Capítulo 4 se presentan las conclusiones y recomendaciones que se obtuvieron al analizar el funcionamiento de la aplicación y los resultados de las pruebas realizadas.

CAPÍTULO 1. MARCO TEÓRICO

1.1. INTRODUCCIÓN

Cada día se implementan más de redes de datos a nivel mundial. A cada momento, miles de personas requieren acceso a Internet, ya sea por negocios, educación, o fines de ocio y entretenimiento. Sin embargo, a medida que las redes crecen se vuelve prioritaria la necesidad de optimizar su funcionamiento.

Los protocolos estándar actualmente utilizados en las redes, permiten la interoperabilidad de dispositivos, así como la globalización de las comunicaciones. Sin embargo, dentro de una organización o empresa específica, muchos de estos protocolos no son necesarios, y solo se los requiere para ciertas tareas específicas.

Es por esto que existe la necesidad de suministrar a los administradores de red, una manera de personalizar sus redes con el fin de mejorar su rendimiento, y optimizar el tratamiento de los datos que manejan.

Dentro de este contexto, las Redes Definidas por Software o *Software Defined Networks* (SDN), presentan una opción para crear soluciones personalizadas para un mejor control y administración de los datos que circulan a través de la red, debido a su facilidad de implementación, capacidad de mejoramiento y adaptabilidad a las circunstancias presentes dentro de las necesidades de la red [1]. Uno de los problemas que se presenta con frecuencia en las redes es la pérdida de paquetes debido a errores en los enlaces que interconectan los distintos nodos.

En este caso, el enfoque tradicional para solucionar este problema se basa en la utilización de protocolos con mecanismos de reenvío de paquetes perdidos desde el terminal que genera la información; sin embargo, esta solución presenta utilización de recursos adicionales en el terminal, ya que se debe considerar la capacidad del mismo para transmitir datos nuevos, así como también datos de reenvío.

El uso de las Redes Definidas por Software como una herramienta para diseñar e implementar un mecanismo de recuperación de datos dentro de una red con errores utilizando el concepto de codificación de red es una alternativa a la solución tradicional a este problema.

La codificación de red se basa en otorgar capacidad de procesamiento de datos a los nodos intermedios de una red, con el fin de permitirles realizar operaciones lógicas sobre los paquetes y gracias a esto implementar un mecanismo para la recuperación de datos perdidos sin necesidad de recurrir a retransmisión por parte del terminal que emite la información.

Esta solución permite generar un mecanismo de recuperación de errores que es transparente para los equipos terminales de la red, permitiéndoles realizar transmisión de información disminuyendo la necesidad de utilizar recursos en retransmisión de paquetes perdidos, lo cual en caso de servidores que responden solicitudes de millones de clientes implica el ahorro de altas cantidades de recursos que se traducen en gastos económicos. A través de un enfoque práctico, este trabajo realiza la implementación de una solución en base a SDN y la muestra como un mecanismo viable en el proceso de recuperación de datos en redes con pérdidas.

1.2. REDES DEFINIDAS POR SOFTWARE

Una SDN se basa en equipos de conectividad que no controlan de manera automática el flujo de paquetes a través de configuraciones prestablecidas, sino que permiten controlar el comportamiento de la red a través de una aplicación externa, alojada en un controlador y que puede ser programada de manera que la red se adapte de la mejor manera posible a los requerimientos del entorno en el que va a funcionar.

En la Figura 1.1 se presenta la arquitectura simplificada de una SDN.

Los dispositivos de red, los cuales pertenecen a la llamada Capa Infraestructura, no poseen un plano de control propio, y por ende dependen directamente del Controlador SDN ubicado en la Capa de Control de la arquitectura, el cual genera reglas de funcionamiento en base a instrucciones establecidas por las aplicaciones personalizadas escritas por el usuario, las cuales se alojan dentro de la Capa Aplicación [1]. La capa aplicación puede alojar una o varias aplicaciones que pueden interactuar con el controlador o con el administrador de red o usuarios tradicionales.

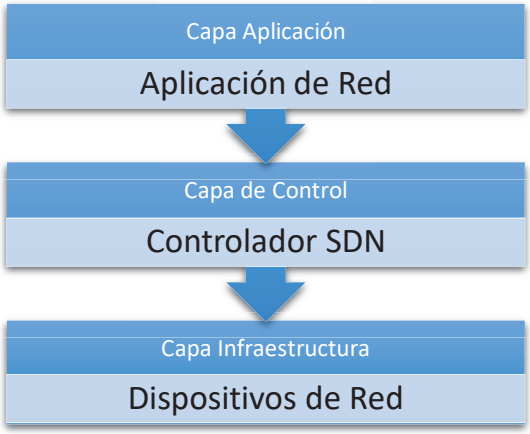


Figura 1.1 Arquitectura de una red SDN

1.3. OPENFLOW

OpenFlow es un protocolo de comunicaciones que provee una interfaz entre el plano de control y los dispositivos de red en una arquitectura SDN a través de un canal seguro.

OpenFlow permite programar los dispositivos de red a través de reglas, las cuales establecen que los dispositivos tomen acciones en base a comparaciones (*match)* de condiciones predefinidas.

Estas reglas pueden especificarse de forma manual por un administrador, o programados para adaptarse de forma automática a las condiciones de la red [3].

A continuación, se presenta un resumen de las características y elementos de la versión 1.3 del protocolo OpenFlow.

1.3.1. TABLAS DE FLUJO

OpenFlow programa las acciones en los dispositivos de red en tablas de flujo. Un dispositivo de red puede tener una o varias tablas, y en cada una de ellas se alojan las reglas, las cuales poseen instrucciones que se ejecutan cuando se genera el *match* de una condición establecida por el controlador SDN.

Cuando un paquete ingresa a un dispositivo de red, se realiza la comparación o *match* de acuerdo a la especificación de cada una de las reglas presentes en la

tabla y en base a prioridades que cada regla posee. Si el *match* es exitoso con una de estas reglas, se procede a ejecutar la instrucción respectiva. Este proceso se resume en la Figura 1.2.

Figura 1.2 Procesamiento de un paquete que ingresa a un dispositivo de red

En caso de no existir un *match* para el paquete, se puede aplicar una regla por defecto (*miss flow entry*) alojado en una tabla de flujo para paquetes sin *match*. Cada entrada en una tabla de flujo contiene:

- **Campos de *match*:** Se utilizan para realizar comparaciones con los paquetes entrantes de acuerdo a campos de cabeceras o metadatos especificados por el controlador SDN o por una tabla previa.
- **Prioridad:** Determina que regla debe emplearse primero en relación al resto.
- **Contadores:** Permiten obtener una estadística de cuantos paquetes han cumplido el *match* de un determinado flujo.
- **Instrucciones:** Determinan la acción a tomar en referencia al paquete.
- **Tiempos de vida:** Mide reglas que han estado presentes durante demasiado tiempo o reglas que no han sido utilizadas después de un número determinado de segundos.
- ***Cookies*:** Son identificadores especiales, denominados opacos, que permiten la identificación de reglas dentro de las tablas de flujo y permiten un uso más amigable de los mismos por parte del controlador o de la aplicación.

1.3.2. MATCH

Cuando un paquete ingresa a un dispositivo de red que utiliza el protocolo OpenFlow (*switch* OpenFlow), el *match* se realiza de acuerdo a la Figura 1.3.

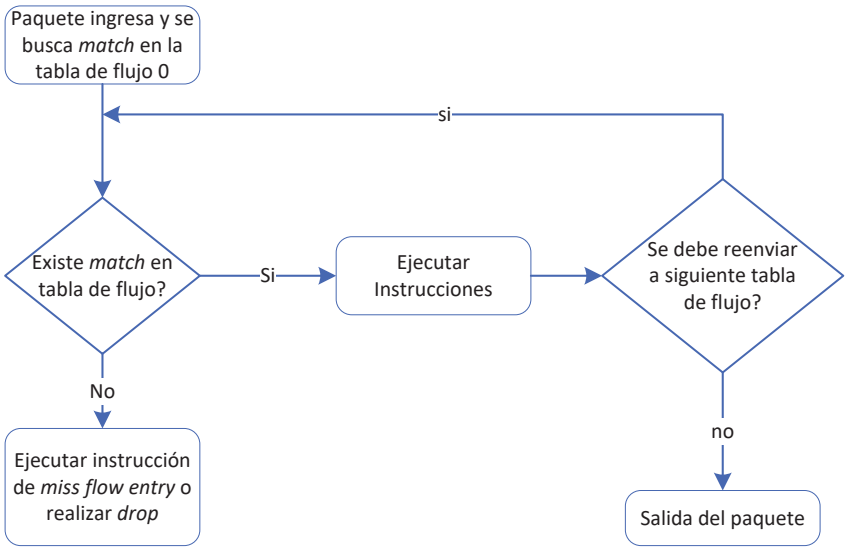


Figura 1.3 Proceso del *match*

En caso de no existir *match* en la primera tabla de flujo, se reenvía el paquete a la siguiente tabla y se busca el *match* es esa tabla.

De la misma manera un paquete puede ser reenviado a buscar el *match* en la siguiente tabla de flujo sí la instrucción ejecutada así lo indica. Cuando un paquete ha ingresado a varias tablas de flujo se dice que ha recorrido un *pipeline* de tablas de flujo [4].

El *match* consiste en buscar si uno o más campos de las distintas cabeceras del paquete (dirección MAC de origen, por ejemplo) coincide con un valor establecido dentro de una de las columnas de la tabla de flujo. Se puede realizar el *match* utilizando campos de cabeceras de protocolos de cada 2 como Ethernet, capa 3 como IP, capa 4 como TCP o UDP, o información referente a las VLAN o etiquetas MPLS, dependiendo de las capacidades del dispositivo de red utilizado. Para mayor información sobre los *matches* se recomienda revisar el Anexo I.

1.3.3. INSTRUCCIONES

Las instrucciones se ejecutan una vez que el paquete que ha ingresado al dispositivo de red ha cumplido con un *match* especifico, estas instrucciones pueden ser:

- ***Meter*** **`meter_id`**: Mide la cantidad de paquetes con un medidor específico (`meter_id`). Si la cantidad máxima de paquetes configurada en el medidor es sobrepasada, el paquete se descarta.
- ***Apply Actions***: Aplica una o varias acciones especificadas dentro de la instrucción de manera inmediata, sin alterarlas o modificarlas.
- ***Clear Actions***: Elimina una o varias acciones de la instrucción especificada.
- ***Write Actions*** **`action(s)`**: Inserta una o varias acciones (`action(s)`) dentro de la instrucción, si la acción existe la sobrescribe, y si no existe la añade al conjunto actual.
- ***Write Metadata***: Escribe metadatos, dentro de un campo asignado dentro de la instrucción. Estos metadatos son usados por el controlador SDN y guardan contadores de distintas propiedades, elementos o estadísticas de la red, del dispositivo, o de los puertos del dispositivo de red.
- ***Goto Table*** **`next_table_id`**: Indica que el paquete sea dirigido a la tabla de flujo con identificador `next_table_id` para posteriores instrucciones.

Los metadatos de OpenFlow 1.3 se encuentran en [3].

Dentro de una tabla de flujo puede existir varias instrucciones con un mismo *match*, en este caso la ejecución depende del protocolo OpenFlow dentro del *switch*.

1.3.4. CONJUNTO DE ACCIONES

Las acciones tienen como objetivo determinar qué sucederá con el paquete. OpenFlow especifica 3 acciones principales que deben ser soportadas por todos los dispositivos de red:

- ***Outport*** **`port_no`**: Reenvía el paquete a través de un puerto especificado determinado por el valor ingresado en el parámetro llamado (`port_no`).
- ***Group*** **`group_id`**: Permite el procesamiento de un paquete por un grupo que puede estar compuesto por varios dispositivos de red o puede estar compuesto de varias tablas de flujo dentro de un mismo dispositivo de red formado una secuencia o *pipeline*. Esta opción se puede utilizar para centralizar la administración de varios dispositivos de red.

- ***Drop***: No es una acción que pueda ser asignada a un *match* determinado, en su lugar, cualquier paquete que no posea una acción de reenvío, o en caso que las instrucciones para el mismo se encuentren vacías, ya sea por no haberlas configurado o haber realizado la instrucción *Clear-Actions*, será eliminado o descartado.

Además, existen acciones adicionales que no son obligatorias como:

- ***Set-Queue*** `queue_id`: Establece una cola para el paquete en base a un identificador (`queue_id`), esto resulta útil en caso de tener una gran cantidad de paquetes reenviados por un puerto particular.
- ***Push-Tag / Pop-Tag ethertype***: Puede añadir o retirar etiquetas a nivel de enlace, como en el caso de VLAN o MPLS.
- ***Set-Field field_tipe value***: Permite modificar campos de las cabeceras de los paquetes. Los campos que se pueden editar son los mismos que se pueden usar en el *match*.
- ***Change-TTL ttl***: permite cambiar los tiempos de vida o TTL de los paquetes IPv4 o IPv6.

Únicamente después de haber realizado todas las acciones diferentes a un *output* se permite el reenvío del paquete por un puerto.

En caso de no existir una opción de reenvío, el paquete se descarta (*drop*).

1.3.5. DATAPATH

Varios dispositivos de red pueden estar conectados a un mismo controlador SDN. Con el fin de distinguir a cada uno de estos dispositivos, dentro de una SDN, cada dispositivo se abstrae a un objeto, conocido como *datapath*.

El *datapath* posee atributos que permiten identificar un dispositivo dentro de la red SDN y conocer sus características y capacidades, además permite al controlador, y a la aplicación, administrar centralizadamente todos los elementos de la red.

La mayoría de los dispositivos de red conectados al controlador SDN son *switches* que funcionan en capa 2 del modelo ISO/OSI, o pueden manejar protocolos de capa 3 como IP o incluso capa 4 como TCP o UDP.

1.3.6. CONEXIÓN SWITCH – CONTROLADOR

La conexión entre el *switch* y el controlador SDN, se realiza a través de una conexión IP. Si el *switch* conoce la dirección IP del controlador, se procede a establecer una conexión TCP o TLS tradicional con el controlador.

Una vez establecida esta conexión, tanto el controlador como el *switch* envían un mensaje tipo `OFPT_HELLO` con el número de versión de OpenFlow más alto que cada uno soporta. La versión a utilizar se selecciona en base al menor número entre la versión recibida y la enviada.

1.3.7. HANDSHAKE

Una vez que ambos extremos han calculado la versión de OpenFlow que soportan en común, y la conexión ha sido establecida, el controlador envía un mensaje tipo `OFPT_FEATURES_REQUEST` solicitando información al dispositivo de red, como su *datapath ID*, el número de tablas de flujo que posee, el número de paquetes que puede enviar al controlador y capacidades y estadísticas que posee, mientras que el dispositivo de red responde con un `OFPT_FEATURES_REPLY` enviando esta información.

1.3.8. CONFIGURACIÓN DEL DISPOSITIVO

El controlador es capaz de establecer o consultar parámetros de configuración al dispositivo de red, a través de mensajes `OFPT_SET_CONFIG` y `OFPT_GET_CONFIG`.

Estos parámetros permiten definir la manera en que los datos serán enviados al controlador, ya sea que se envíen todos los bytes del paquete, un fragmento del mismo, o solo los metadatos del mismo.

1.3.9. MODIFICACIÓN DE TABLAS DE FLUJO

Una vez que se ha establecido la conexión entre el dispositivo de red y el controlador, el controlador puede enviar mensajes al dispositivo de red para añadir, modificar o borrar reglas de sus tablas, estos mensajes pueden ser:

- **`ADD`**: Añade una nueva regla a una tabla establecida, si existe una regla igual en la tabla del dispositivo, esta se reemplaza, actualizando así sus tiempos de vida.
- **`MODIFY`**: Edita o modifica una regla existente dentro de una tabla, esto permite actualizar los valores de las cookies que la identifican, contadores o tiempos de vida, así como también editar instrucciones y acciones para un *match* establecido.
- **`DELETE`**: Elimina una regla de una tabla seleccionada, siempre y cuando en esta no exista paquetes en proceso haciendo uso de esta regla.

1.4. RYU

Ryu es un controlador SDN, basado en Python y con unas API (*Application Programing Interfaces*) bien definidas que permiten la creación e implementación de aplicaciones de control de red.

Ryu soporta múltiples protocolos de comunicación entre dispositivos de red y el controlador, entre ellos OpenFlow [5].

Dos módulos de Ryu permiten el manejo de todos los mensajes especificados por el protocolo OpenFlow versión 1.3, estos son:

- **`ofproto_v1_3`**: Contiene todas las definiciones de objetos para manejo de mensajes OpenFlow.
- **`ofproto_v1_3_parser`**: Codifica o decodifica los mensajes OpenFlow, tanto para ser enviados a los dispositivos o para ser interpretados por Ryu.

1.5. MININET

Mininet es un simulador de red que permite crear y probar topologías de red en tiempo real.

Una de sus principales ventajas es trabajar con Open vSwitch, una aplicación de Linux capaz de simular un dispositivo de red, conectarlo con un controlador, y comunicarse con el mismo a través del protocolo OpenFlow [6].

Mininet es administrado a través del CLI (*Command Line Interface)* de Linux, a través de la cual, se pueden cargar o crear topologías, especificar el controlador a utilizar, e incluso administrar *hosts* virtuales, enlaces y dispositivos intermedios.

Existen dos métodos para crear topologías en Mininet:

- Directamente, al momento de iniciar Mininet con el comando `mn`, pasando como argumentos del mismo las características de la topología y enlaces.
- A través de un *script* escrito en lenguaje Python en el cual se especifican detalles de la topología e incluso parámetros de simulación.

Mininet tiene la capacidad de crear *hosts* virtuales, los cuales se presentan como consolas de usuario separadas dentro de la máquina principal. Estos *hosts* pueden ejecutar comandos simples e incluso generar tráfico que pueda ser enviado a través de la red virtual, pueden ser asignados direcciones IP, tomar direccionamiento automático vía DHCP, e incluso configurar su dirección MAC física. Una manera más completa de conectar *hosts* a Mininet, es a través de asociaciones entre puertos virtuales de un Open vSwitch e interfaces de red físicas, esto permite conectar otros equipos o máquinas virtuales a la topología de simulación, permitiendo una mayor funcionalidad.

1.6. RED CON TOPOLOGÍA TIPO BUTTERFLY

Esta topología de red consiste en 6 nodos distribuidos de manera tal que existen 7 enlaces entre los mismos, y los nodos extremos de la red (nodos **s1**, **s2**, **s3** y **s4**) están conectados a los equipos terminales. Esta topología se muestra en la Figura 1.4 y permite la implementación de codificación de red de manera simple, para entender su funcionamiento y ventajas.

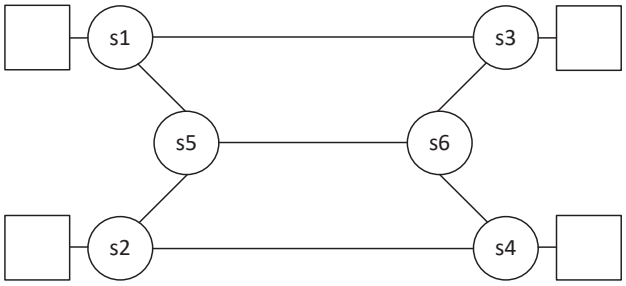


Figura 1.4 Topología tipo *Butterfly*

1.7. CODIFICACIÓN DE RED

Acorde a varios autores, el término Codificación de Red o *Network Coding* se originó en [7], por lo que es un campo de estudio relativamente joven, cuya definición, aplicaciones y ventajas aún se encuentran en desarrollo.

En la primera publicación realizada sobre el tema, *Network Coding* se define como "codificación en un nodo de una red", en donde el término "codificación" se deja abierto para cualquier tipo de manejo de datos de entrada y salida en el nodo. Es decir, los nodos intermedios de la red no solo reenvían los datos, sino que además los procesan para realizar tareas específicas [2].

Es así que *Network Coding* permite dar un nivel de procesamiento de datos a los nodos de la red, con el fin de mejorar el funcionamiento de la misma.

1.8. EJEMPLO DE CODIFICACIÓN DE RED

A continuación, se presentan dos escenarios: en el primero se explica el funcionamiento de una red tradicional mientras que en el segundo escenario se usa codificación de red [7].

1.8.1. ESCENARIO SIN CODIFICACIÓN DE RED

Se asume el siguiente escenario inicial:

- Un equipo terminal fuente de datos *Server*, conectado al nodo **s1** y al nodo **s2** de la red, desea enviar paquetes **p1** y **p2** a los equipos terminales *Client1* y *Client2*, los cuales deben recibir ambos paquetes para conocer el mensaje completo. La topología básica de esta red se muestra en la Figura 1.5.

Figura 1.5 Escenario sin Codificación de Red

- El paquete **p1** se envía al nodo **s1** y al nodo **s2**.
- Los paquetes son reenviados a los nodos **s3** y **s4** para posteriormente ser entregados a sus respectivos destinos, como se muestra en la Figura 1.6.

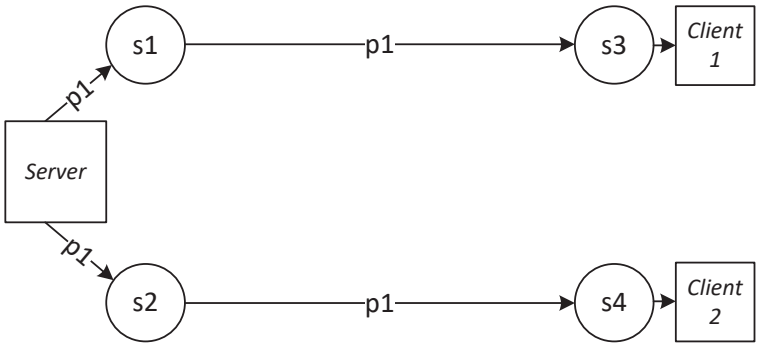


Figura 1.6 Fuente de datos *Server* envía el paquete **p1** a *Client1* y *Client2*

- Una vez entregado el primer paquete, se realiza en envío del paquete **p2** utilizando el mismo procedimiento.
- Cada terminal puede ahora unir ambos paquetes y tener el mensaje completo enviado por *Server,* como se aprecia en la Figura 1.7.

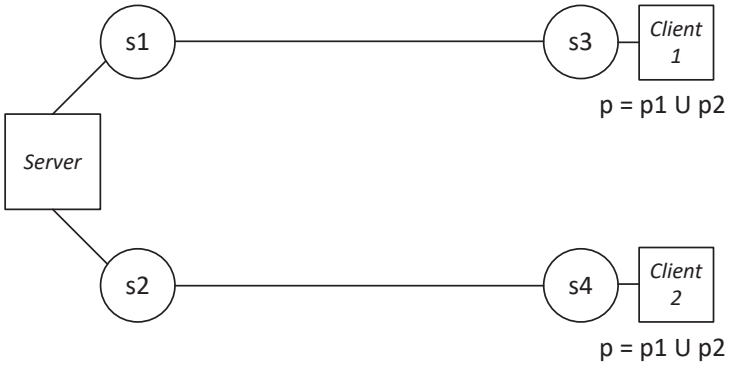


Figura 1.7 *Client1* y *Client2* obtienen el mensaje **p** uniendo **p1** y **p2**

En este escenario se puede contemplar que es necesario que la fuente *Server* envíe dos veces cada paquete, lo cual implica la existencia de redundancia en la transmisión.

1.8.2. ESCENARIO CON CODIFICACIÓN DE RED

- La topología se cambia a una de tipo *butterfly*, con dos nodos adicionales, **s5** y **s6** como se aprecia en la Figura 1.8.

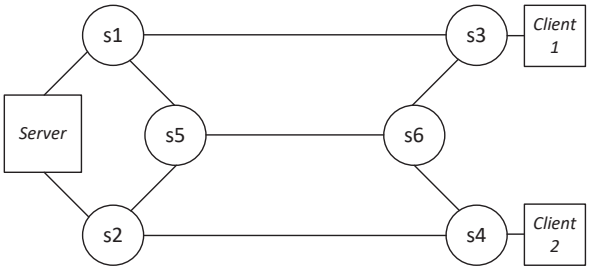


Figura 1.8 Escenario con Network Coding

- La fuente *Server* envía el paquete **p1** al nodo **s1** y el paquete **p2** al nodo **s2**. Los nodos **s1** y **s2** reenvían los paquetes a los nodos **s3** y **s4**, y reenvían una copia de **p1** y **p2** al nodo **s5**, el cual generará un mensaje nuevo a través de la suma binaria de los dos paquetes recibidos. Este nuevo mensaje **p3**, es enviado al nodo **s6**. El nodo **s6** envía **p3** a los nodos **s3** y **s4,** como se ve en la Figura 1.9.

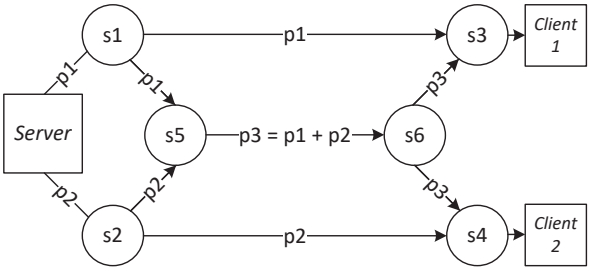


Figura 1.9 Proceso de envío de paquetes con Codificación de Red

- Una vez que los nodos **s3** y **s4** reciban el paquete **p1** y **p2,** respectivamente, pueden usar **p3** para obtener el paquete faltante y generar el mensaje completo, sin necesidad de que la fuente realice un segundo envío, lo cual se ve en la Figura 1.10.

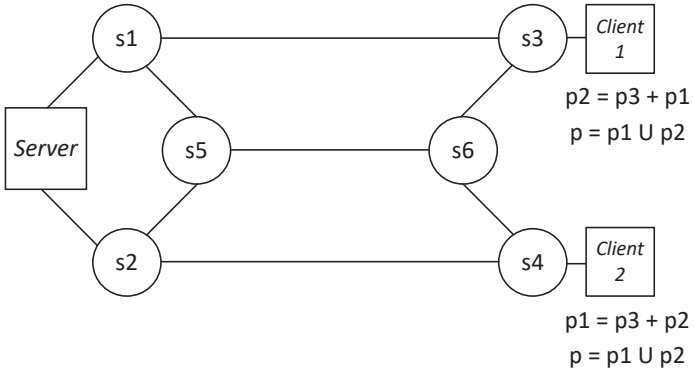


Figura 1.10 Recuperación del paquete **p** usando **p3**

El ejemplo presentado permite contemplar varias ventajas, entre ellas:

- **Eficiencia**: Permite la trasmisión de información usando menos cantidad de paquetes enviados desde la fuente. En el ejemplo se aprecia cómo se redujo la transmisión de paquetes en un 50%.
- **Robustez**: Menos paquetes circulando por la red significa una menor probabilidad de perder un paquete o incluso que los paquetes perdidos puedan ser recuperados en los nodos intermedios sin necesidad de enviar una solicitud de retransmisión a la fuente.
- **Seguridad**: En el ejemplo presentado, un intruso escuchando y capturando paquetes en el enlace entre el nodo 1 y 3, es ahora incapaz de conocer el mensaje completo, ya que no conoce el segundo fragmento, y el mensaje completo ha sido reconstruido y entregado por otro segmento de la red.

1.9. CODIFICACIÓN DE RED PARA RECUPERACIÓN DE DATOS

A partir de este punto y durante el resto del presente trabajo, se usará el escenario descrito a continuación:

- La red *butterfly* tiene dos fuentes *Server1* y *Server2*. La fuente *Server1* se comunica con el terminal *Client1* y la fuente *Server2* con el terminal *Client2*.
- Se asume que los enlaces entre los nodos **s1** y **s3**, y los nodos **s2** y **s4** tienen pérdidas, pudiendo perder paquetes que circulen por los mismos con una tasa mayor a cero (*error rate > 0*).
- El resto de enlaces no posee pérdidas, es decir que su tasa de error es igual a cero (*error rate = 0*).
- El enlace entre **s5** y **s6** es más lento que el resto de enlaces, es decir posee retardos mucho mayores en la transmisión de paquetes que el resto de enlaces, en este caso se utiliza un retardo de por lo menos 10 veces el valor del retardo en el resto de enlaces (10**delay(1) < delay(2))*. Este retardo garantiza que a pesar de las distintas trayectorias que toman nos paquetes, el enlace entre **s5** y **s6** será siempre más lento y por lo tanto los paquetes que circulen por el mismo llegarán al final.

La topología descrita se aprecia en la Figura 1.11.

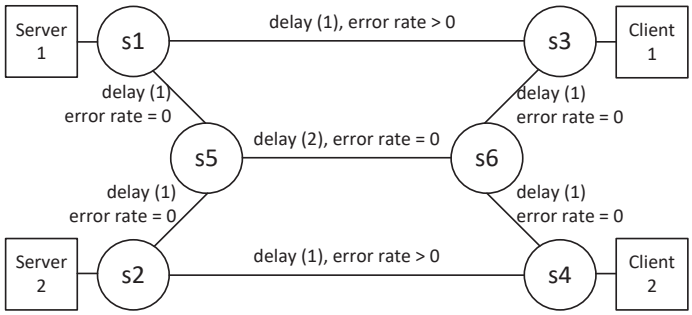


Figura 1.11 Escenario para análisis

- Se realizará el proceso de codificación en el nodo **s5**, el cual toma los paquetes que proceden de los nodos **s1** y **s2**, y genera un nuevo paquete **p3** a través de la suma binaria de **p1** y **p2**
- El paquete **p3** se reenvía al nodo **s6**.
- Los nodos **s3** y **s4** reciben paquetes desde los nodos **s1** y **s2**, y los reenvían a los terminales *Client1* y *Client2* respectivamente, así como también al nodo **s6**.
- El nodo **s6** recibe los paquetes que ingresan a los nodos **s3** y **s4** y que provienen de los nodos **s1** y **s2**, así como también el paquete **p3** enviado por el nodo **s5**. Este proceso de intercambio de información en la red se puede observar en la Figura 1.12.

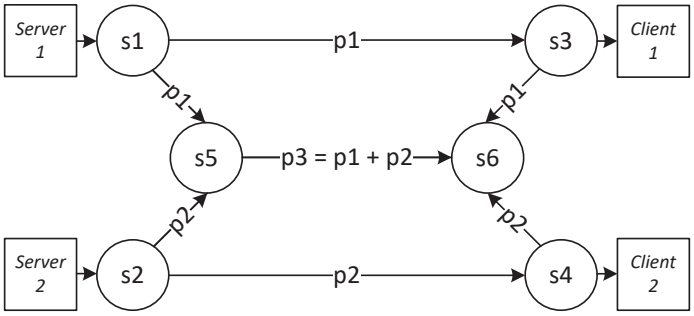


Figura 1.12 Flujo de paquetes

- Debido a la diferencia de velocidad de los enlaces, el paquete **p3** llegará al nodo **s6** después de que **p1** y **p2** hayan llegado. Por lo tanto, el nodo **s6** puede tomar una de las siguientes acciones:

a) Si recibe el paquete **p3**, y se ha recibido tanto **p1** como **p2**, entonces se descarta **p3** puesto que no es necesario recuperar el paquete. Esto se aprecia en la Figura 1.13.

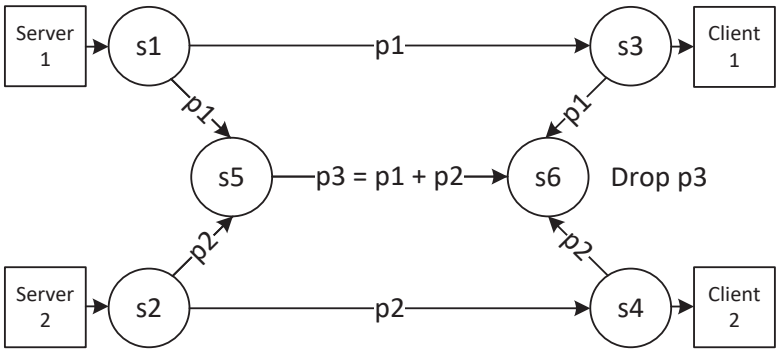


Figura 1.13 Escenario de paquetes sin pérdidas

b) Si recibe el paquete **p3**, pero solo se ha recibido uno de los paquetes, sea **p1** o **p2**, se reconstruye el paquete faltante usando **p3** y se lo reenvía al nodo correspondiente para ser entregado a su destino. La recuperación de **p1** se observa en la Figura 1.14.

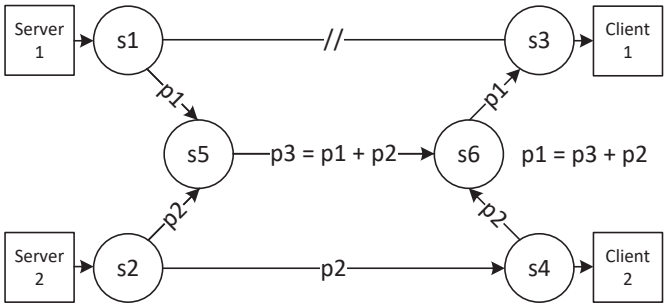


Figura 1.14 Escenario de **p1** perdido y recuperado en el nodo **s6**

c) Si recibe el paquete **p3**, y no ha recibido ningún otro paquete adicional, descarta **p3** y no realiza ninguna otra acción. En este caso la recuperación de paquetes deberá hacerse con reenvío.

Este escenario presenta las siguientes ventajas:

- Se brinda cierto nivel de capacidad a la red para recuperar paquetes perdidos sin necesidad de que los terminales o fuentes tengan conocimiento de esta pérdida.
- La recuperación automática de paquetes permite que los terminales usen protocolos que no utilizan reenvío, como UDP, con una menor probabilidad de tener paquetes perdidos.
- En el caso de TCP, la cantidad de reenvíos disminuye de manera notable, debido a que la probabilidad de pérdida disminuye.

Sin embargo, se presenta también algunas posibles desventajas:

- Se aumenta el nivel de procesamiento en los nodos **s5** y **s6**, así como también se incluye nuevas reglas de reenvío para los nodos **s1**, **s2**, **s3**, y **s4** lo cual aumenta el retardo de la red.
- La recuperación de paquetes no funciona en el caso de que ambos paquetes se pierdan, y el paquete de codificación generado en esa situación no cumple ningún propósito.

CAPÍTULO 2. DISEÑO E IMPLEMENTACIÓN

En este capítulo se presenta la pregunta de investigación que el presente trabajo pretende contestar, los objetivos que el ambiente desarrollado cumple y el alcance que posee, así como también la metodología que se utiliza. Posteriormente se presenta el resultado del diseño e implementación de los distintos elementos del ambiente de red: la configuración de las 7 máquinas virtuales implementadas; la red virtual con topología de red; las aplicaciones para el controlador Ryu; y las aplicaciones para las tareas de codificación y decodificación de tramas de datos.

2.1. PREGUNTA DE INVESTIGACIÓN

¿Es la codificación de red, un mecanismo viable para la implementación de una técnica de recuperación de paquetes utilizando una Red Definida por Software?

2.2. OBJETIVO GENERAL

Implementar una aplicación para emplear codificación de red en una red SDN.

2.3. OBJETIVOS ESPECÍFICOS

- Analizar los principios de la codificación de red y su aplicación en redes de datos que tienen pérdida de paquetes.
- Implementar una red SDN con topología tipo *butterfly* para el desarrollo de pruebas de la aplicación que correrá sobre el controlador SDN.
- Implementar un mecanismo de codificación de información y de recuperación de datos en el servidor controlador SDN y analizar los resultados obtenidos a partir de las pruebas realizadas.

2.4. ALCANCE

En este trabajo se implementó una aplicación de codificación de red, para lo cual se configuraron 7 máquinas virtuales dentro de una máquina física, se levantó un

ambiente de red virtual en Mininet y se escribieron cuatro aplicaciones en lenguaje Python.

Las 7 máquinas virtuales se configuraron utilizando el *software* de virtualización VMWare Workstation 12, usando el sistema operativo Linux Ubuntu 16.04, y se les asignó nombres de acuerdo a cada una de las funciones que realizan dentro del escenario implementado, así: **NC Master**, **NC Host1**, **NC Host2**, **NC Host3**, **NC Host4**, **NC Switch5** y **NC Switch6**, cuyas funciones se detallan a continuación.

NC Master es la máquina virtual central del entorno de red; ejecuta la red virtual en Mininet, el controlador Ryu y las aplicaciones que generan las reglas de funcionamiento a ser instaladas en cada uno de los *switches* virtuales de la red.

La red virtual se construye utilizando Mininet a través de un archivo escrito en lenguaje de programación Python llamado `RedTopologiaButterfly.py`. Dentro de este archivo existe una clase llamada `Butterfly` la cual define 6 *switches* virtuales denominados **s1**, **s2**, **s3**, **s4**, **s5** y **s6** e interconectados de acuerdo a una topología tipo *butterfly*, poseen enlaces con tiempos de retardo específicos y tasas de pérdidas configurables a través del archivo de configuración `Perdidas.ini`. El resto de máquinas virtuales (**NC Host1**, **NC Host2**, **NC Host3**, **NC Host4, NC Switch5** y **NC Switch6**) se encuentran conectadas a **NC Master** y a su vez a la red virtual, de acuerdo a lo presentado en la Figura 2.1.

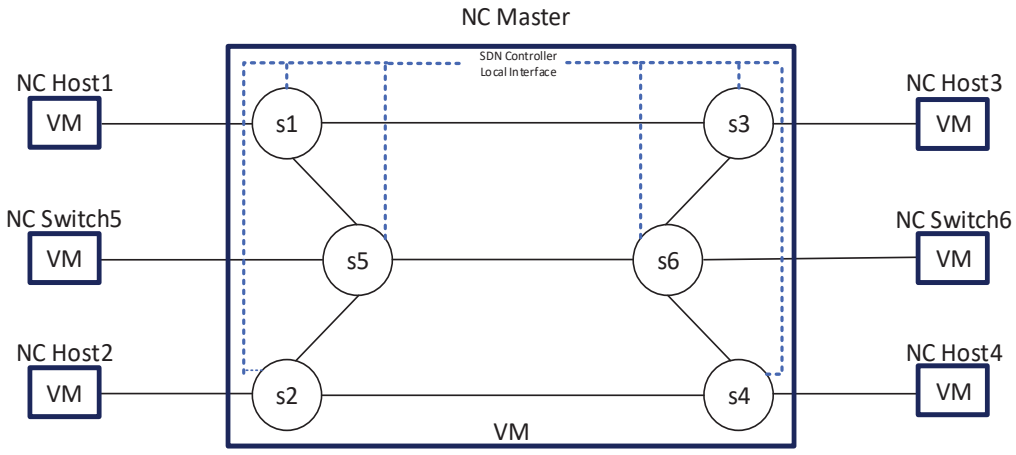


Figura 2.1 Entorno de red implementado

NC Host1, **NC Host2**, **NC Host3** y **NC Host4** funcionan como equipos terminales, y únicamente envían y reciben tramas de datos. **NC Switch5** y **NC Switch6**

ejecutan aplicaciones que realizan el proceso de codificación de red a través de una suma binaria de las tramas enviadas por parte de los terminales.

Cada uno de los *switches* virtuales de la red se conecta a un controlador Ryu alojado dentro de **NC Master** a través de la interfaz de red local. El controlador Ryu puede recibir las reglas de funcionamiento desde una de las dos aplicaciones escritas en el lenguaje de programación Python desarrolladas para este trabajo:

- La primera aplicación se encuentra definidas en el archivo `ReglasTransmisionNormal.py` y contiene reglas que permiten la transmisión de tramas entre **NC Host1** y **NC Host3** a través del enlace entre **s1** y **s3**, y la transmisión de tramas entre **NC Host2** y **NC Host4** a través del enlace entre **s2** y **s4**.
- La segunda aplicación se encuentra dentro del archivo `ReglasCodificacionRed.py` y contiene las reglas de la primera aplicación y reglas adicionales para reenviar todas las tramas a **NC Switch5** y **NC Switch6** a través de los enlaces entre **s1** y **s2** con **s5**, y los enlaces entre **s3** y **s4** con **s6**. Contiene además las reglas para transmitir las tramas producto del proceso de codificación de red a través del enlace entre **s5** y **s6.**

NC Switch5 ejecuta la aplicación definida en el archivo llamado `CodificadorRedSwitch5.py` y **NC Switch6** ejecuta la aplicación definida en el archivo llamado `CodificadorRedSwitch6.py`.

Ambas aplicaciones están escritas en lenguaje de programación Python y se encargan de capturar tramas generadas en cada uno de los equipos terminales (**NC Host1**, **NC Host2**, **NC Host3** y **NC Host4**), y realizar la codificación de red, tanto para generar tramas nuevas, producto de la codificación de red, así como para recuperar tramas perdidas.

Se realizan pruebas de funcionamiento en el entorno de red donde se envían paquetes utilizando el protocolo ICMP y el comando `ping` y se obtienen resultados a partir de las mismas.

Utilizando Wireshark se capturan las tramas enviadas y recibidas en cada uno de los terminales, y con esta información se determina el número de tramas que se pierden en los enlaces entre **s1** y **s3** y entre **s2** y **s4**. Se calcula el porcentaje de

pérdidas obtenido tanto para la transmisión sin utilizar codificación de red, así como para la transmisión utilizando codificación de red. En base a la diferencia entre los porcentajes de pérdidas obtenidos para ambos casos, se calcula el porcentaje de recuperación debido a la aplicación de codificación de red.

Dentro de este trabajo no se pretende analizar la diferencia de tiempos ni los retardos introducidos por la utilización de codificación de red, así como tampoco se pretende analizar el impacto de las tramas adicionales generadas producto de la codificación de red [2].

2.5. METODOLOGÍA

El presente trabajo cumple con los siguientes aspectos:

- **Enfoque:** Presenta un enfoque cuantitativo, se generan estadísticas de tramas perdidas y recuperadas en el escenario propuesto y se obtienen los porcentajes de tramas recuperadas aplicando codificación de red.
- **Tipo de Investigación:** Se realiza de manera experimental de la siguiente manera:
    - Se implementa una aplicación para codificación de red compuesta de una red virtual con topología tipo *Butterfly*, aplicaciones para el controlador SDN Ryu, aplicaciones para realizar el proceso de codificación y recuperación de tramas y máquinas virtuales que alojen los distintos elementos o funcionen como terminales para generar datos a transmitir.
    - Se especifican los parámetros de la tasa de pérdidas en los enlaces de la red virtual, y después se realiza la transmisión de tramas entre los equipos terminales de la red. Las tramas enviadas y recibidas por cada uno de los terminales son capturadas y analizadas para obtener conclusiones referentes a la utilización de la codificación de red.
- **Técnica de Recolección de Información:** La recolección de información se realiza mediante pruebas de funcionamiento, en las cuales se genera un número predefinido de tramas que son inyectadas en la red, la cual puede tener enlaces con pérdidas.

- Las tramas enviadas y recibidas por cada uno de los terminales son capturadas y analizadas con el fin de determinar la cantidad de tramas perdidas, recuperadas y transmitidas.
- **Procedimiento empleado para la Obtención y Análisis de la Información:** La recolección de datos que permita determinar la capacidad de la codificación de red para recuperar tramas perdidas se realiza a través de datos obtenidos a partir de las pruebas realizadas de la siguiente manera:
    - Se establece una tasa de pérdidas entre 0 y 100% en los enlaces de la red virtual que comunican a los terminales de la red.
    - Se ejecuta las distintas aplicaciones que componen el entorno de pruebas como la red virtual, el controlador Ryu con las aplicaciones que generan las reglas de funcionamiento y las aplicaciones para codificación y recuperación de tramas.
    - Una vez levantado el entorno se genera un intercambio de paquetes ICMP entre los terminales de la red virtual. A partir de cada terminal se obtienen los datos del número de tramas transmitidas y recibidas, utilizando el capturador de tramas Wireshark.
    - El análisis de la información obtenida se realiza con la diferencia entre el número de tramas totales trasmitidas y el número de tramas recibidas para calcular el porcentaje de pérdidas para la prueba.
    - En base a la diferencia entre el porcentaje de pérdidas obtenido en la red de datos normal y el porcentaje obtenido con la implementación de codificación de red se calcula el porcentaje de recuperación de la red y se generan las conclusiones correspondientes.

2.6. COMPONENTES DE LA APLICACIÓN IMPLEMENTADA

En primer lugar, se presentan las características de las máquinas virtuales utilizadas, sus funciones dentro del entorno de red y las aplicaciones que ejecutan.

Después se presenta el archivo escrito en el lenguaje Python `RedTopologiaButterfly.py`, el cual crea una red con topología tipo *butterfly*.

La red cumple con las características de retardos y pérdidas necesarios.

También se expone una aplicación que correrá sobre el controlador denominada `ReglasTransmisionNormal.py` que permite generar reglas para transmisión sin codificación de red y otra aplicación llamada `ReglasCodificacionRed.py` que permite generar reglas para transmisión con codificación de red las cuales se ejecutan sobre el controlador RYU, este controlador controla a los *switches* de la red virtual implementada en Mininet. También se muestran las aplicaciones que proveen los mecanismos de codificación y recuperación de datos denominadas `CodificadorRedSwitch5.py` y `CodificadorRedSwitch6.py`. Finalmente se presenta un resumen de las distintas pruebas realizadas para distintos escenarios de interés.

Se aclara que en este trabajo no se presentan los procedimientos de instalación de las distintas herramientas utilizadas, ya que los mismos han sido cubiertos por anteriores trabajos y pueden encontrarse en [8] y [9].

2.6.1. MÁQUINAS VIRTUALES

En este proyecto se utilizaron un total de 7 máquinas virtuales, cada una con funciones específicas detalladas a continuación:

- 1 máquina virtual llamada **NC Master**, donde se ejecuta el ambiente de red definido en el archivo `RedTopologiaButterfly.py` mediante Mininet, y el controlador Ryu sobre el cual se ejecutan las aplicaciones de los archivos `ReglasTransmisionNormal.py` y `ReglasCodificacionRed.py`.
- 4 máquinas virtuales llamadas **NC Host1**, **NC Host2**, **NC Host3** y **NC Host4** conectadas a **NC Master**, y que funcionan como equipos terminales.
- 2 máquinas virtuales llamadas **NC Switch5** y **NC Switch6** conectadas a **NC Master**. En **NC Switch5** se ejecuta la aplicación `CodificadorRedSwitch5.py` y en **NC Switch6** se ejecuta la aplicación `CodificadorRedSwitch6.py`. Ambas aplicaciones son responsables de la codificación y decodificación de datos para el proceso de recuperación de tramas: la primera (`CodificadorRedSwitch5.py`) se encarga de codificar tramas de **NC Host1** y **NC Host2** y decodificar tramas de **NC Host3** y **NC Host4**, mientras que la segunda (`CodificadorRedSwitch6.py`) se encarga

de codificar tramas de **NC Host3** y **NC Host4** y decodificar tramas de **NC Host1** y **NC Host2**.

Las 7 máquinas virtuales poseen las siguientes características:

- Sistema Operativo Linux Ubuntu 16.04 de 32 bits
- 2 procesadores con un núcleo lógico cada uno
- 2 GB de memoria RAM
- Disco duro virtual de almacenamiento dinámico de hasta 100 GB

Para la conexión de las máquinas virtuales a la red, se han que:

- Cada interfaz de red posee una dirección MAC predefinida por el usuario, con el fin de facilitar la administración y control del ambiente de red.
- **NC Master** posee 6 interfaces de red que sirven para conectarse con cada una de las demás máquinas virtuales.
- El resto de máquinas virtuales poseen una interfaz de red, cada una de ellas conectada a una interfaz de red específica de **NC Master**.
- El direccionamiento IP se realiza haciendo uso de 6 redes distintas, con el fin de evitar comunicación entre todas las máquinas virtuales.

Un resumen de las direcciones MAC e IP utilizadas en cada interfaz de red de cada máquina virtual se presenta en la Tabla 2.1.

Para interconectar las máquinas virtuales entre sí se emplea el asistente de red de VMWare.

Un esquema de la interconexión de las máquinas virtuales realizado se muestra en la Figura 2.2.

Tabla 2.1. Direccionamiento de máquinas virtuales

Máquina Virtual	Interfaz	Dirección MAC	Dirección IP
NC Master	ens33	00:00:00:00:00:01	192.168.10.10/24
	ens34	00:00:00:00:00:02	192.168.20.20/24
	ens35	00:00:00:00:00:03	192.168.30.30/24
	ens36	00:00:00:00:00:04	192.168.40.40/24
	ens37	00:00:00:00:00:05	192.168.50.50/24
	ens38	00:00:00:00:00:06	192.168.60.60/24
NC Host1	ens33	00:00:00:00:00:11	192.168.10.11/24
NC Host2	ens33	00:00:00:00:00:22	192.168.20.22/24
NC Host3	ens33	00:00:00:00:00:33	192.168.30.33/24
NC Host4	ens33	00:00:00:00:00:44	192.168.40.44/24
NC Switch5	ens33	00:00:00:00:00:55	192.168.50.55/24
NC Switch6	ens33	00:00:00:00:00:66	192.168.60.66/24

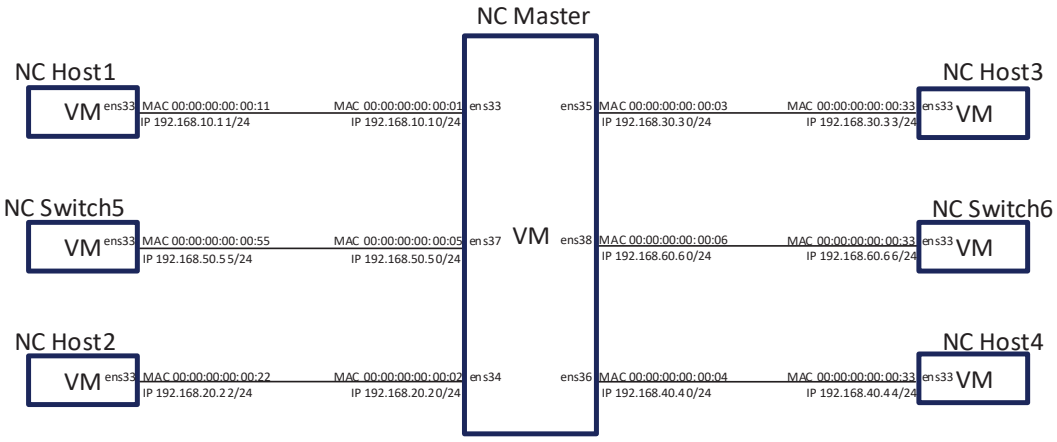


Figura 2.2 Conexión de máquinas virtuales

2.6.2. RED CON TOPOLOGÍA TIPO BUTTERFLY

La Figura 2.3 presenta la red con topología tipo *butterfly* que se implementa mediante el archivo de Python `RedTopologíaButterfly.py`, la cual consta de 6 nodos llamados **s1**, **s2**, **s3**, **s4**, **s5** y **s6**. Cada nodo es un *switch* virtual y posee un puerto asociado a cada una de las interfaces de red de **NC Master**. Cada *switch* se conecta mediante la interfaz local de **NC Master** con el controlador Ryu, y cada enlace posee un tiempo de retardo específico (*delay*). Los enlaces entre **s1** y **s3** y entre **s2** y **s4** poseen además una tasa de pérdida de tramas (*loss rate*) que puede variar entre 0% y 100%.

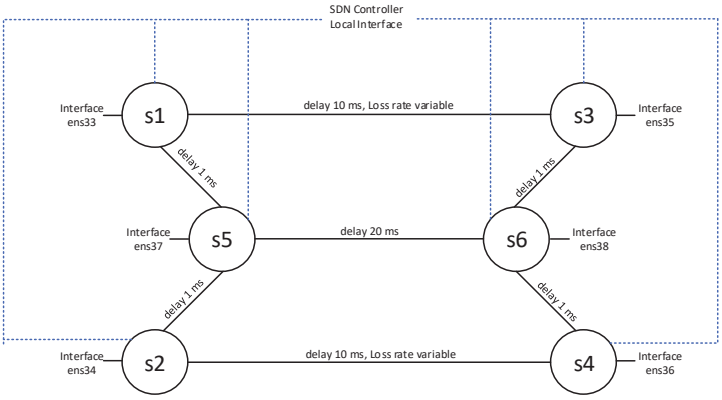


Figura 2.3 Red con topología tipo *Butterfly*

Esta red virtual se implementa usando las librerías disponibles en Mininet. A continuación, se describe como está conformado este archivo.

2.6.2.1. Librerías Utilizadas

El código necesario para importar las librerías requeridas se muestra en la Figura 2.4.

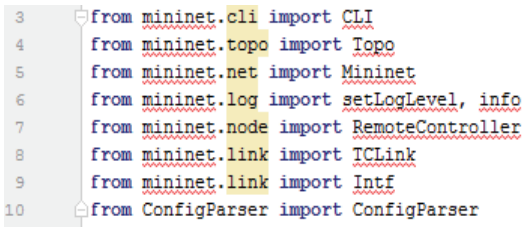


Figura 2.4 Librerías importadas, archivo `RedTopologíaButterfly.py`

En la línea 3 se importa `CLI` que permitirá la comunicación con la terminal de Linux, con el cual se podrá ejecutar comandos del *shell*, *scripts* o escribir mensajes en la consola.

En la línea 4 se importa `Topo` que permitirá la creación de topologías personalizadas, esta librería es obligatoria para crear una nueva clase que instancie un objeto de topología creado por el usuario.

En la línea 5 se importa `Mininet` que permitirá la utilización del entorno de simulación, es decir permite ejecutar o detener la simulación de una red virtual con topología personalizada de manera automática, así como también ejecutar comandos de Mininet durante la simulación.

En la línea 6 se importa `setLogLevel` que permite establecer el nivel de información a presentar, e `info` que despliega la información en pantalla.

En la línea 7 se importa `RemoteController` que permitirá la conexión a un controlador externo.

En la línea 8 se importa `TCLink` que permitirá crear enlaces con características personalizadas, como ancho de banda, retardos o tasa de pérdida de tramas.

En la línea 9 se importa `Intf` que permitirá la asociación de puertos virtuales creados por Mininet con las interfaces de red de la máquina virtual donde se ejecuta la aplicación, lo que permite conectar esta red con clientes externos.

En la línea 10 se importa `ConfigParser` que permite obtener valores almacenados en archivos de configuración.

2.6.2.2. Configuración de Pérdidas

Con el propósito de definir los porcentajes de pérdidas se crea el archivo `Perdidas.ini` mostrado en la Figura 2.5. en el cual se definen los valores de pérdidas que se desea configurar en los enlaces entre **s1** y **s3**, y entre **s2** y **s4**. Además, se muestra el código necesario para abrir el archivo de configuración e importar los valores de pérdidas.

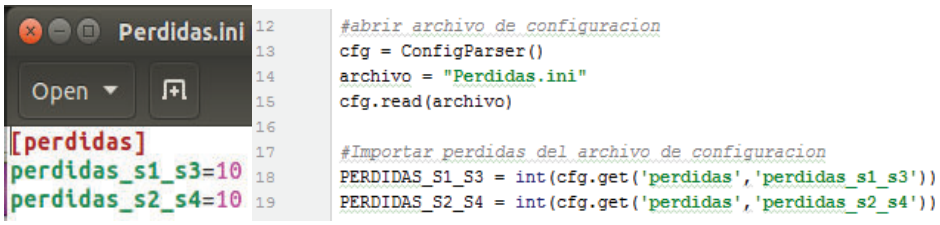


Figura 2.5 Archivo de configuración `Perdidas.ini` (izquierda) y código para abrir archivo de configuración e importar valores de pérdidas (derecha)

2.6.2.3. Clase para crear la Topología *Butterfly*

Esta clase llamada `Butterfly` (ver Figura 2.6) hereda sus propiedades de la clase `Topo` de Mininet y define el método `__init__()` (línea 24) en donde se crean los *switches* y enlaces entre los mismos, estableciendo la estructura de la red *butterfly*.

```
21  #Clase para Topología Butterfly
22  class Butterfly( Topo ):
23
24      def __init__( self ):
25
26          # Inicializar la Topología
27          Topo.__init__( self )
28
29          # Crear Switches
30          S1 = self.addSwitch('s1')
31          S2 = self.addSwitch('s2')
32          S3 = self.addSwitch('s3')
33          S4 = self.addSwitch('s4')
34          S5 = self.addSwitch('s5')
35          S6 = self.addSwitch('s6')
36
37          self.addLink(S1, S3, port1=2, port2=2, bw=100, delay='10ms', loss=PERDIDAS_S1_S3)
38          self.addLink(S2, S4, port1=2, port2=2, bw=100, delay='10ms', loss=PERDIDAS_S2_S4)
39          self.addLink(S1, S5, port1=3, port2=1, bw=100, delay='1ms')
40          self.addLink(S2, S5, port1=3, port2=2, bw=100, delay='1ms')
41          self.addLink(S3, S6, port1=3, port2=1, bw=100, delay='1ms')
42          self.addLink(S4, S6, port1=3, port2=2, bw=100, delay='1ms')
43          self.addLink(S5, S6, port1=3, port2=3, bw=100, delay='20ms')
```

Figura 2.6 Clase `Butterfly`

Como se puede observar en las líneas 30 a la 35 se crean los 6 *switches* necesarios para la red *butterfly*. En las líneas 37 a la 43 se procede a crear los enlaces entre cada uno de los *switches* de acuerdo a la Figura 2.2. Estos enlaces tienen las siguientes características:

- Tasa de transmisión de 100 Mbps, para lo cual se utiliza el parámetro `bw`.
- Retardo de trasmisión expresado en milisegundos, para lo cual se utiliza el parámetro `delay`. Estos retardos permiten controlar que las tramas que viajan por los enlaces entre **s1** y **s3** y entre **s2** y **s4** lleguen a su destino antes (retardo de 10 ms) que aquellos que viajan por el enlace entre **s5** y **s6** (retardo de 20ms). El resto de enlaces posee un retardo de 1 milisegundo, el mínimo permitido por Mininet.
- Los enlaces entre **s1** y **s3** y entre **s2** y **s4** poseen pérdidas definidas por el parámetro `loss` establecido a partir del archivo de configuración `Perdidas.ini`. La tasa de pérdidas puede variar entre el 0 y 100%. El resto de enlaces no poseen pérdidas.

2.6.2.4. Creación del ambiente de red

El ambiente de red se ejecuta a través de una secuencia de instrucciones de Python, descrito a continuación:

Se crea un objeto llamado `net`, el cual es una instancia de Mininet que contendrá el ambiente virtual de red. Este ambiente recibe como argumentos: la topología a utilizar, en este caso una instancia de la clase `Butterfly`; el controlador a utilizar, en este caso se usa un controlador remoto; y el tipo de enlaces a utilizar, que son tipo `TCLink`, lo cual permite establecer retardos, tasas de error y tasas de transmisión predefinidos. Este código se presenta en la Figura 2.7.

```
50    #Crear instancia de Mininet
51    net = Mininet(topo=Butterfly(), controller=RemoteController('c0', ip='127.0.0.1'), link=TCLink)
```

Figura 2.7 Creación de instancia `net`

Se asocia cada una de las interfaces de red de **NC Master** con puertos de los *switches* virtuales definidos dentro de la clase `Butterfly()`, permitiendo así conectar el resto de máquinas virtuales a la red virtual. Para esto se utiliza la clase `Intf`. El código para esta asociación se presenta en la Figura 2.8.

Para asociar a cada *switch* se emplea un arreglo contenido en la clase `net` llamado `switches`, en el cual el *switch* **s1** se encuentra en la posición 0 del arreglo, el *switch* **s2** se encuentra en la posición 1 y así sucesivamente.

```
#Asociacion de interfaces de red con puertos de los switches virtuales
Intf('ens33', net.switches[0], port=1)
info('*** Adding hardware interface', 'ens33', 'to switch', net.switches[0].name, '\n')
Intf('ens34', net.switches[1], port=1)
info('*** Adding hardware interface', 'ens34', 'to switch', net.switches[1].name, '\n')
Intf('ens35', net.switches[2], port=1)
info('*** Adding hardware interface', 'ens35', 'to switch', net.switches[2].name, '\n')
Intf('ens36', net.switches[3], port=1)
info('*** Adding hardware interface', 'ens36', 'to switch', net.switches[3].name, '\n')
Intf('ens37', net.switches[4], port=4)
info('*** Adding hardware interface', 'ens37', 'to switch', net.switches[4].name, '\n')
Intf('ens38', net.switches[5], port=4)
info('*** Adding hardware interface', 'ens38', 'to switch', net.switches[5].name, '\n')
```

Figura 2.8 Asociación de interfaces de red con puertos de *switches* virtuales

Al final del archivo se inicia la simulación mediante el objeto `net` y mediante la clase `CLI` el ambiente virtual se asocia a la terminal de Linux con el fin de mostrar en pantalla información relacionada con el inicio de la simulación, creación de dispositivos virtuales, enlaces y propiedades, así como también permitir la ejecución de comandos de Mininet en tiempo de ejecución. El método `stop()` se utiliza para eliminar todas las instancias y terminar la ejecución. Estas últimas instrucciones se muestran en la Figura 2.9.

El archivo completo que contiene el ambiente de red se incluye en el Anexo II.

```
#Inicio, ejecución y finalización del ambiente virtual de red
net.start()
CLI(net)
net.stop()
```

Figura 2.9 Inicio, ejecución y finalización del ambiente virtual de red

El ambiente final tiene la estructura mostrada en la Figura 2.10, donde se presenta el número de identificación de cada puerto de los *switches* virtuales. Los enlaces presentados en azul son los utilizados para la conexión al controlador SDN, el cual se ejecuta dentro de **NC Master**.

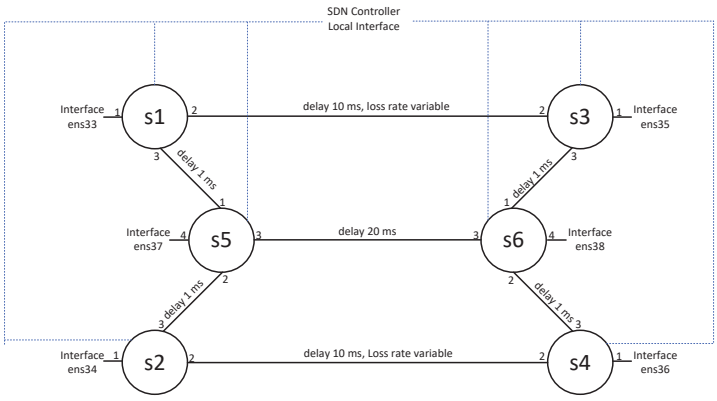


Figura 2.10 Red virtual creada por Mininet

Como se mostró en la Figura 2.1, cada interfaz de red de **NC Master** se encuentra conectada a una máquina virtual por lo que el entorno de red completo corresponde al presentado en la Figura 2.11. Para facilitar la visualización de información se presenta solo los 2 últimos octetos de la dirección MAC de cada interfaz de red.

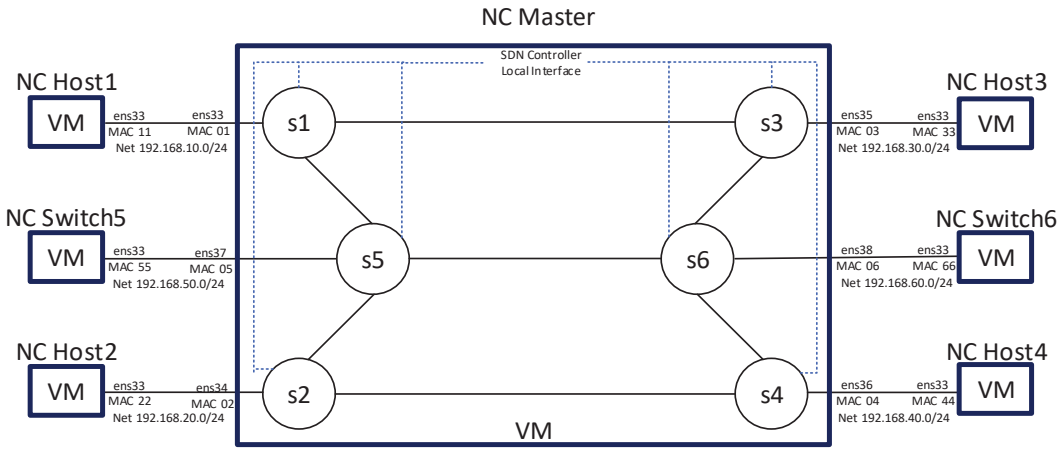


Figura 2.11 Entorno de red completo

2.6.3. APLICACIONES PARA EL CONTROLADOR RYU

Para la comparación entre una red con codificación de red frente a una red tradicional, se implementaron dos aplicaciones que correrán sobre el controlador Ryu instalado en **NC Master**: una cuyas reglas representan a una red tradicional donde los dispositivos intermedios solo reenvían las tramas por los puertos de salida correspondiente y no realizan ningún tipo de tarea adicional, y otra con las reglas necesarias para permitir el envío de tráfico a las aplicaciones que realizarán la codificación y recuperación de tramas. Se debe considerar que en ambas aplicaciones la comunicación entre dispositivos se encuentra restringida de la siguiente manera: **NC Host1** solo se puede comunicar con **NC Host3** y **NC Host2** únicamente se puede comunicar con **NC Host4**. Además, no se permite el paso de tramas cuyo destino es la dirección MAC de *broadcast* (FF:FF:FF:FF:FF:FF), con lo cual se requiere que los equipos terminales **NC Host1**, **NC Host2**, **NC Host3** y **NC Host4** posean tablas ARP predefinidas por el usuario.

Todo esto para facilitar la implementación y verificación del funcionamiento de la codificación de red.

2.6.3.1. Librerías Utilizadas

Ambas aplicaciones importan las librerías de acuerdo al código presentado en la Figura 2.12:

```
1  from ryu.base import app_manager
2  from ryu.controller import ofp_event
3  from ryu.controller.handler import CONFIG_DISPATCHER
4  from ryu.controller.handler import set_ev_cls
5  from ryu.ofproto import ofproto_v1_3
6  from ConfigParser import ConfigParser
```

Figura 2.12 Librerías importadas

En la línea 1 se importa `app_manager` que es la clase principal de toda aplicación para Ryu, esta clase permite a la aplicación correr sobre el controlador y comunicarse con el mismo para asignarle tareas.

En la línea 2 se importa `ofp_event` que es la clase que maneja todos los eventos definidos por Ryu, de acuerdo al funcionamiento del protocolo OpenFlow.

En la línea 3 se importa `CONFIG_DISPATCHER` que se utilizará para ejecutar un método que permita instalar reglas a cada uno de los *switches* durante la fase de configuración del dispositivo.

En la línea 4 se importa `set_ev_cls` que permitirá la captura de eventos para la ejecución de tareas.

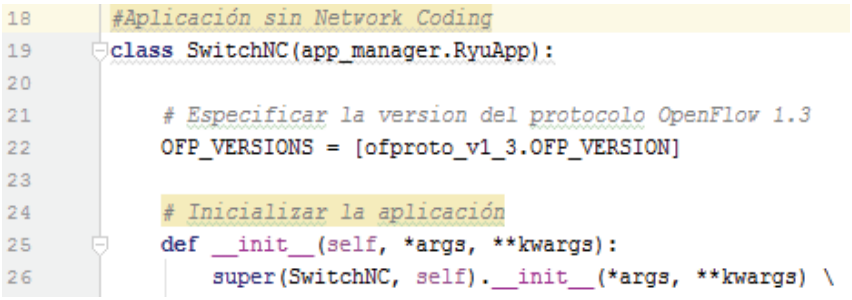
En la línea 5 se importa `ofproto_v1_3`, la cual dispone de métodos que permiten generar tramas con formato OpenFlow versión 1.3.

En la línea 6 se importa `ConfigParser` para el manejo de archivos de configuración.

Ambas aplicaciones definen una clase llamada `SwitchNC`, la cual hereda sus propiedades de la clase `app_manager.RyuApp`, como se muestra en la línea 19 del código presentado en la Figura 2.13.

En la línea 22 se especifica la versión de OpenFlow a utilizar en la aplicación.

El método `__init__()` definido en la línea 25 se ejecuta al instanciar un objeto de la clase durante la ejecución del controlador.



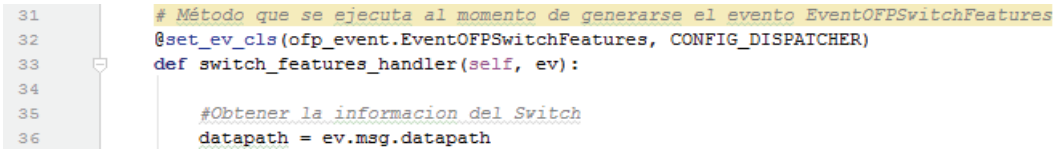
```
18  #Aplicación sin Network Coding
19  class SwitchNC(app_manager.RyuApp):
20
21      # Especificar la version del protocolo OpenFlow 1.3
22      OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
23
24      # Inicializar la aplicación
25      def __init__(self, *args, **kwargs):
26          super(SwitchNC, self).__init__(*args, **kwargs) \
```

Figura 2.13 Inicio de las aplicaciones

La Figura 2.14 muestra la siguiente sección de código, la cual se ejecuta una vez inicializada la aplicación. Ambas aplicaciones utilizan el decorador `@set_ev_cls` mostrado en la línea 32.

Este decorador indica que el método `switch_features_handler()`, definido en la línea 33, se ejecutará al momento de producirse el evento `EventOFPSwitchFeatures`, el cual se genera cuando un *switch* envía sus características al controlador a la espera de recibir reglas a instalar de acuerdo a lo establecido por `CONFIG_DISPATCHER`.

La información del evento se captura en el objeto `ev`, dentro del cual se encuentra un objeto llamado `msg` el cual contiene el *datapath* del *switch* que ha generado el evento. Este *datapath* se obtiene en la línea 36.



```
31      # Método que se ejecuta al momento de generarse el evento EventOFPSwitchFeatures
32      @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
33      def switch_features_handler(self, ev):
34
35          #Obtener la informacion del Switch
36          datapath = ev.msg.datapath
```

Figura 2.14 Método `switch_features_handler()` y obtención del *datapath*

2.6.3.2. Aplicación sin Codificación de Red - ReglasTransmisionNormal.py

Esta aplicación genera reglas que reenvían información por un puerto de salida en base a los parámetros: puerto de entrada, dirección MAC de origen y dirección MAC de destino.

En la Tabla 2.2 se presenta un resumen de las reglas que esta aplicación envía al controlador para ser instaladas en los distintos *switches* de acuerdo a su *datapath ID.*

Tabla 2.2 Reglas para Aplicación sin Codificación de Red

SWITCH	*DATAPATH ID*	CONDICIONES DE *MATCH*			ACCIÓN
		PUERTO DE ENTRADA	DIRECCIÓN MAC ORIGEN	DIRECCIÓN MAC DESTINO	PUERTO DE SALIDA
s1	1	1	00:00:00:00:00:11	00:00:00:00:00:33	2
		2	00:00:00:00:00:33	00:00:00:00:00:11	1
s2	2	1	00:00:00:00:00:22	00:00:00:00:00:44	2
		2	00:00:00:00:00:44	00:00:00:00:00:22	1
s3	3	1	00:00:00:00:00:33	00:00:00:00:00:11	2
		2	00:00:00:00:00:11	00:00:00:00:00:33	1
s4	4	1	00:00:00:00:00:44	00:00:00:00:00:22	2
		2	00:00:00:00:00:22	00:00:00:00:00:44	1

Las direcciones MAC para las reglas de esta aplicación se obtienen a través del archivo de configuración como se muestra en la Figura 2.15.

```
8   cfg = ConfigParser()
9   archivo ="MACaddressRYU.ini"
10  cfg.read(archivo)
11
12  #Importar direcciones MAC del archivo de configuracion
13  MAC_HOST1 = cfg.get('macs_sinNC','mac_host1')
14  MAC_HOST2 = cfg.get('macs_sinNC','mac_host2')
15  MAC_HOST3 = cfg.get('macs_sinNC','mac_host3')
16  MAC_HOST4 = cfg.get('macs_sinNC','mac_host4')
```

Figura 2.15 Direcciones MAC desde el archivo de configuración

Para instalar estas reglas en cada uno de los *switches* de la red virtual se debe crear un *match*, acciones e instrucciones. Estos tres elementos se deben codificar en un mensaje para que el controlador lo envíe al *switch*. Mediante el método `addFlow()` y con base en el *datapath ID* recibido en el objeto `ev` dentro del método `switch_features_handler()` se envían los parámetros necesarios para instalar las reglas especificadas en la Tabla 2.2. La Figura 2.16 muestra el código para generar las reglas en el *switch* **s1** con *datapath ID* 1.

```
38      #Reglas para s1
39      if datapath.id == 1:
40
41          self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
42                       in_port=1, out_port=2, parser=ev.msg.datapath.ofproto_parser,
43                       datapath=ev.msg.datapath,
44                       ofproto=ev.msg.datapath.ofproto)
45
46          self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
47                       in_port=2, out_port=1, parser=ev.msg.datapath.ofproto_parser,
48                       datapath=ev.msg.datapath,
49                       ofproto=ev.msg.datapath.ofproto)
```

Figura 2.16 Parámetros para las reglas en **s1**

Para realizar estas tareas en la aplicación se define el método `addFlow()`.

Este método dispone de argumentos que permiten pasar la información de la regla a instalar (línea 118), como prioridad (`priority`), dirección MAC de origen (`eth_src`), dirección MAC de destino (`eth_dst`), puerto de entrada (`in_port`), puerto de salida (`out_port`), y la información del codificador (`parser`), *datapath* (`datapath`) y protocolo OpenFlow (`ofproto`); esta información permite crear el correspondiente *match* (línea 120), las acciones (línea 121) e instrucciones (línea 122) y genera el mensaje (línea 123) para que el controlador realice el envío (línea 124) al *switch* correspondiente, como se muestra en la Figura 2.17:

```
117     # Instalar las reglas en los Switches
118     def addFlow(self, priority, eth_src, eth_dst, in_port, out_port, parser, datapath, ofproto):
119
120         match = parser.OFPMatch(in_port=in_port, eth_src=eth_src, eth_dst=eth_dst)
121         actions = [parser.OFPActionOutput(out_port)]
122         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
123         mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match, instructions=inst)
124         datapath.send_msg(mod)
```

Figura 2.17 Método `addFlow()`

El código completo de esta aplicación se encuentra en el Anexo III.

2.6.3.3. Aplicación con Codificación de Red - ReglasCodificacionRed.py

Esta aplicación es más compleja que la anterior, y utiliza la capacidad de OpenFlow para permitir editar cabeceras de las tramas de datos, así como también asigna reglas a los *switches* **s5** y **s6**.

Considerando el funcionamiento de la codificación de red para recuperación de datos presentada en el Capítulo 1, se establece que:

- Los *switches* **s1, s2, s3** y **s4** reciben tramas procedentes de **NC Host1**, **NC Host2**, **NC Host3** y **NC Host4** respectivamente y deben reenviarlos a los *switches* **s5** y **s6** para que se realice el proceso de codificación.
- Todas las tramas que son recibidas por los *switches* **s5** y **s6** deben ser codificadas; sin embargo, ya que este nivel de procesamiento supera las capacidades de los *switches,* se utilizan las aplicaciones externas instaladas en **NC Switch5** y **NC Switch6** que permitirán simular dispositivos de red con una mejor capacidad de procesamiento de información.

- Además, para que **NC Switch5** y **NC Switch6** puedan recibir estas tramas y no desecharlas, se debe alterar la dirección MAC de destino de cada trama enviada a estos equipos con la dirección MAC adecuada correspondiente a cada interfaz de red de **NC Switch5** y **NC Switch6** conectada a la red virtual (00:00:00:00:00:55 y 00:00:00:00:00:66).
- Las tramas generadas como resultado del proceso de codificación de red tendrán direcciones MAC nuevas (00:00:00:00:00:77 para las codificadas en el equipo **NC Switch5** y 00:00:00:00:00:88 para las codificadas en el equipo **NC Switch6**) con el fin de reconocerlas fácilmente dentro de la red virtual y las aplicaciones de codificación de red.
- Estas tramas viajarán por el enlace entre **s5** y **s6** de la red *butterfly* para ser utilizadas en la recuperación de datos de ser necesario. De acuerdo a lo analizado en el Capítulo 1, únicamente se puede realizar codificación de red si existe intercambio de tramas entre **NC Host1** y **NC Host3** y entre **NC Host2** y **NC Host4** de manera simultánea.
- En caso de que esto no sea posible, se deben instalar reglas para enviar tramas sin utilizar codificación de red y sin generar pérdidas.
- Para identificar este tipo de tramas de manera simple dentro de la red se utilizan direcciones MAC distintas, especificadas en el archivo de configuración:
    - Para tramas de **NC Host1** se utiliza 00:00:00:00:01:11.
    - Para tramas de **NC Host2** se utiliza 00:00:00:00:02:22.
    - Para tramas de **NC Host3** se utiliza 00:00:00:00:03:33.
    - Para tramas de **NC Host4** se utiliza 00:00:00:00:04:44.

Con el fin de facilitar la visualización de las reglas que el controlador deberá instalar en base a las trayectorias deseadas para cada paquete, se presenta el diagrama de secuencia en la Figura 2.18. Este diagrama muestra la trayectoria que seguirán dos tramas de datos: una trama **p1**, enviada desde **NC Host1** a **NC Host3**, y una trama **p2**, enviada desde **NC Host2** a **NC Host4**.

Ambas tramas generan una trama codificada realizando una operación XOR o suma binaria entre **p1** y **p2**. Esta nueva trama ingresa a la red y no es entregada a ningún *host*.

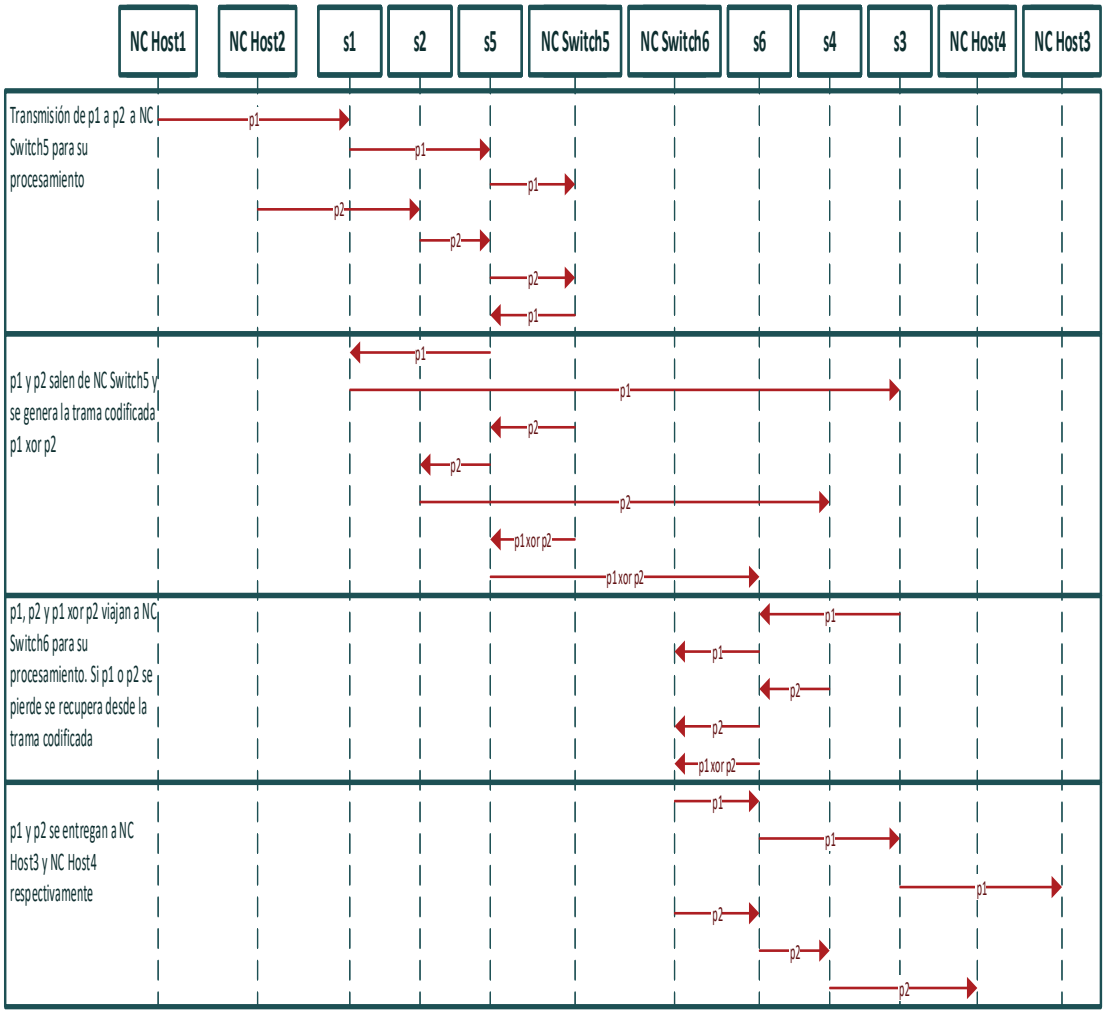


Figura 2.18 Diagrama de secuencia

En base al diagrama de secuencia de la Figura 2.18 se conoce la trayectoria que los paquetes deberán seguir dentro de la red *butterfly* y en base a esta información se generan las reglas necesarias para cumplir estos requerimientos, facilitando la evaluación de la red.

Se aclara que para paquetes trasmitidos desde **NC Host3** para **NC Host1** y paquetes trasmitidos desde **NC Host4** para **NC Host2**, la secuencia es la misma, pero en orden inverso.

En la Tabla 2.3 se presenta un resumen de las reglas que el controlador deberá instalar, en donde se incluye además las reglas necesarias para trasmitir tramas sin necesidad de usar codificación de red en caso de ser necesario:

Tabla 2.3 Reglas para Aplicación con Codificación de Red

SWITCH	*DATAPATH ID*	*MATCH*			ACCIONES	
		PUERTO DE ENTRADA	DIRECCIÓN MAC ORIGEN	DIRECCIÓN MAC DESTINO	NUEVA DIRECCIÓN MAC DESTINO	PUERTO DE SALIDA
s1	1	1	00:00:00:00:00:11	00:00:00:00:00:33	No	3
		2	00:00:00:00:00:33	00:00:00:00:00:11	No	3
		3	00:00:00:00:00:11	00:00:00:00:00:33	No	2
		3	00:00:00:00:00:33	00:00:00:00:00:11	No	1
s2	2	1	00:00:00:00:00:22	00:00:00:00:00:44	No	3
		2	00:00:00:00:00:44	00:00:00:00:00:22	No	3
		3	00:00:00:00:00:22	00:00:00:00:00:44	No	2
		3	00:00:00:00:00:44	00:00:00:00:00:22	No	1
s3	3	1	00:00:00:00:00:33	00:00:00:00:00:11	No	3
		2	00:00:00:00:00:11	00:00:00:00:00:33	No	3
		3	00:00:00:00:00:33	00:00:00:00:00:11	No	2
		3	00:00:00:00:00:11	00:00:00:00:00:33	No	1
s4	4	1	00:00:00:00:00:44	00:00:00:00:00:22	No	3
		2	00:00:00:00:00:22	00:00:00:00:00:44	No	3
		3	00:00:00:00:00:44	00:00:00:00:00:22	No	2
		3	00:00:00:00:00:22	00:00:00:00:00:44	No	1
s5	5	1	00:00:00:00:00:11	00:00:00:00:00:33	00:00:00:00:00:55	4
		2	00:00:00:00:00:22	00:00:00:00:00:44	00:00:00:00:00:55	4
		1	00:00:00:00:00:33	00:00:00:00:00:11	00:00:00:00:00:55	4
		2	00:00:00:00:00:44	00:00:00:00:00:22	00:00:00:00:00:55	4
		3	00:00:00:00:00:77	00:00:00:00:00:88	00:00:00:00:00:55	4
		4	00:00:00:00:00:11	00:00:00:00:00:33	No	1
		4	00:00:00:00:00:22	00:00:00:00:00:44	No	2
		4	00:00:00:00:00:33	00:00:00:00:00:11	No	1
		4	00:00:00:00:00:44	00:00:00:00:00:22	No	2
		4	00:00:00:00:00:88	00:00:00:00:00:77	No	3
s6	6	1	00:00:00:00:00:11	00:00:00:00:00:33	00:00:00:00:00:66	4
		2	00:00:00:00:00:22	00:00:00:00:00:44	00:00:00:00:00:66	4
		1	00:00:00:00:00:33	00:00:00:00:00:11	00:00:00:00:00:66	4
		2	00:00:00:00:00:44	00:00:00:00:00:22	00:00:00:00:00:66	4
		3	00:00:00:00:00:88	00:00:00:00:00:77	00:00:00:00:00:66	4
		4	00:00:00:00:00:11	00:00:00:00:00:33	No	1
		4	00:00:00:00:00:22	00:00:00:00:00:44	No	2
		4	00:00:00:00:00:33	00:00:00:00:00:11	No	1
		4	00:00:00:00:00:44	00:00:00:00:00:22	No	2
		4	00:00:00:00:00:77	00:00:00:00:00:88	No	3
Reglas para tramas donde no se aplica codificación de red						
s1	1	3	00:00:00:00:01:11	00:00:00:00:00:33	No	2
		2	00:00:00:00:03:33	00:00:00:00:00:11	No	3
s2	2	3	00:00:00:00:02:22	00:00:00:00:00:44	No	2
		2	00:00:00:00:04:44	00:00:00:00:00:22	No	3
s3	3	3	00:00:00:00:03:33	00:00:00:00:00:11	No	2
		2	00:00:00:00:01:11	00:00:00:00:00:33	No	3
s4	4	3	00:00:00:00:04:44	00:00:00:00:00:22	No	2
		2	00:00:00:00:02:22	00:00:00:00:00:44	No	3
s5	5	4	00:00:00:00:01:11	00:00:00:00:00:33	No	1
		4	00:00:00:00:02:22	00:00:00:00:00:44	No	2
		1	00:00:00:00:03:33	00:00:00:00:00:11	No	4
		2	00:00:00:00:04:44	00:00:00:00:00:22	No	4
s6	6	1	00:00:00:00:01:11	00:00:00:00:00:33	No	4
		2	00:00:00:00:02:22	00:00:00:00:00:44	No	4
		4	00:00:00:00:03:33	00:00:00:00:00:11	No	1

Las direcciones MAC para las reglas de esta aplicación se obtienen a través de un archivo de configuración, en el código que se muestra en la Figura 2.19.

Para este caso se requiere las direcciones MAC de cada equipo conectado a la red, las direcciones MAC para identificar a las tramas que no se pueden utilizar en el proceso de codificación de red y las direcciones MAC de las tramas generadas por codificación de red. Todas estas direcciones se conocen con anterioridad y pueden ser cambiadas en caso de conectar un dispositivo distinto.

```
12  #Importar direcciones MAC del archivo de configuracion
13  MAC_HOST1 = cfg.get('macs_conNC','mac_host1')
14  MAC_HOST2 = cfg.get('macs_conNC','mac_host2')
15  MAC_HOST3 = cfg.get('macs_conNC','mac_host3')
16  MAC_HOST4 = cfg.get('macs_conNC','mac_host4')
17  MAC_SWITCH5 = cfg.get('macs_conNC','mac_switch5')
18  MAC_SWITCH6 = cfg.get('macs_conNC','mac_switch6')
19  MAC_HOST1_U = cfg.get('macs_conNC','mac_host1_u')
20  MAC_HOST2_U = cfg.get('macs_conNC','mac_host2_u')
21  MAC_HOST3_U = cfg.get('macs','mac_host3_u')
22  MAC_HOST4_U = cfg.get('macs_conNC','mac_host4_u')
23  MAC_H3XORH4_77 = cfg.get('macs_conNC','mac_h3xorh4_77')
24  MAC_H1XORH2_88 = cfg.get('macs_conNC','mac_h1xorh2_88')
```

Figura 2.19 Direcciones MAC desde el archivo de configuración

El método `switch_features_handler()` envía los parámetros necesarios para instalar las reglas especificadas en la Tabla 2.3. La Figura 2.20 muestra el código para generar las reglas en el *switch* **s1** (*datapath ID* 1). El resto de *switches* se configuran de la misma manera, de acuerdo a las reglas correspondientes.

```
46      # Reglas para s1
47      if datapath.id == 1:
48
49          self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
50                       in_port=1, out_port=3, parser=ev.msg.datapath.ofproto_parser,
51                       datapath=ev.msg.datapath,
52                       ofproto=ev.msg.datapath.ofproto)
53
54          self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
55                       in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
56                       datapath=ev.msg.datapath,
57                       ofproto=ev.msg.datapath.ofproto)
58
59          self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
60                       in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
61                       datapath=ev.msg.datapath,
62                       ofproto=ev.msg.datapath.ofproto)
63
64          self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
65                       in_port=3, out_port=1, parser=ev.msg.datapath.ofproto_parser,
66                       datapath=ev.msg.datapath,
67                       ofproto=ev.msg.datapath.ofproto)
68
69          self.addFlow(priority=0, eth_src=MAC_HOST1_U, eth_dst=MAC_HOST3,
70                       in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
71                       datapath=ev.msg.datapath,
72                       ofproto=ev.msg.datapath.ofproto)
73
74          self.addFlow(priority=0, eth_src=MAC_HOST3_U, eth_dst=MAC_HOST1,
75                       in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
76                       datapath=ev.msg.datapath,
77                       ofproto=ev.msg.datapath.ofproto)
```

Figura 2.20 Parámetros para reglas en **s1**

De la misma manera que en la aplicación anterior, en esta aplicación define el método `addFlow()` para crear los elementos necesarios para las reglas. Debido a que existen distintos tipos de reglas, este método varía en su implementación, como se ve en la Figura 2.21.

```
383     # Instalar las reglas en los Switches
384     def addFlow(self, priority, eth_src, eth_dst, in_port, out_port, parser, datapath, ofproto):
385
386         #Regla a instalar en s1, s2, s3 o s4
387         if datapath.id in [1, 2, 3, 4]:
388             actions = [parser.OFPActionOutput(out_port)]
389
390         # Regla a instalar en s5 o s6
391         if datapath.id in [5, 6]:
392
393             # Regla si s5 o s6 envia a NC Switch5 o NC Switch6
394             if out_port == 4:
395
396                 if datapath.id == 5:
397                     new_mac_dst = MAC_SWITCH5
398
399                 if datapath.id == 6:
400                     new_mac_dst = MAC_SWITCH6
401
402                 actions = [parser.OFPActionSetField(eth_dst=new_mac_dst), parser.OFPActionOutput(out_port)]
403
404             # Regla si s5 o s6 recibe desde NC Switch5 o NC Switch6
405             if in_port == 4:
406                 actions = [parser.OFPActionOutput(out_port)]
407
408         match = parser.OFPMatch(in_port=in_port, eth_src=eth_src, eth_dst=eth_dst)
409         inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
410         mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match, instructions=inst)
411         datapath.send_msg(mod)
```

Figura 2.21 Método `addFlow()`

En este método se realiza lo siguiente:

- Si la regla se va a instalar en **s1, s2, s3** o **s4** la acción consiste únicamente en reenviar la trama por el puerto de salida especificado (líneas 387 y 388).
- Si la regla se va a instalar en **s5** o **s6** existen dos casos. Si la trama se envía a **NC Switch5** o **NC Switch6** se establece una nueva dirección MAC y la acción consiste en modificar la dirección MAC de destino de la trama antes de enviarla. Por el contrario, si la trama se recibe desde **NC Switch5** o **NC Switch6** se reenvía la trama por el puerto de salida especificado sin modificación alguna (líneas 391 a 406).

Una vez reconocido el tipo de acciones a instalar, el procedimiento de creación de *match*, instrucción, codificación y envío se realiza de la misma manera que en la aplicación anterior. Además, en este caso se incluye las reglas para los *switches* **s5** y **s6**, los cuales no requerían reglas para la aplicación anterior**.**

El código completo de esta aplicación se encuentra en el Anexo IV.

2.6.4. APLICACIONES PARA CODIFICACIÓN DE RED: CODIFICADORREDSWITCH5.PY Y CODIFICADORREDSWITCH6.PY

Estas aplicaciones realizan las siguientes tareas: reciben tramas procedentes de cada uno de los equipos terminales **NC Host1**, **NC Host2**, **NC Host3** y **NC Host4**;

las almacenan y generar nuevas tramas producto del proceso de codificación de red, reenvían las tramas de vuelta a la red y transmiten las tramas codificadas a través del enlace entre **s5** y **s6**; detectan si una trama se ha perdido y en base a la trama codificada y una trama normal pueden recuperar la trama perdida y reenviarla para su entrega a su destino; finalmente, identifican si existen tramas que se han enviado y no pueden ser utilizadas para generar tramas codificadas las reenvían para ser entregadas a su destino.

La aplicación desarrollada para el proceso de codificación de red depende de la máquina virtual en la que se ejecuta: la aplicación `CodificadorRedSwitch5.py` se ejecuta en **NC Switch5** y la aplicación `CodificadorRedSwitch6.py` se ejecuta sobre **NC Switch6**. Ambas importan librerías de acuerdo al segmento de código presentado en la Figura 2.22.

```
1  import socket
2  import os
3  import threading
4  from time import clock
5  import ConfigParser
```

Figura 2.22 Librerías importadas por las Aplicaciones de Codificación de Red

En la línea 1 se importa `socket`, la cual permite crear *sockects* para la recepción y envío de tramas a través de la interfaz de red de la máquina virtual.

En la línea 2 se importa `os`, la cual permite la ejecución de comandos de Linux.

En la línea 3 se importa `threading`, la cual permite la creación de hilos adicionales en la aplicación.

En la línea 4 se importa `clock`, que permite el manejo del reloj del sistema.

En la línea 5 se importa `ConfigParser` para el manejo de archivos de configuración.

Ambas aplicaciones importan las direcciones MAC desde el archivo de configuración `MACAddress.ini`, como se muestra en la Figura 2.23. Se importan las direcciones MAC de cada máquina virtual, las direcciones MAC que identifican tramas codificadas en **NC Switch5** o en **NC Switch6** y las direcciones MAC a utilizar para identificar tramas que no pueden ser enviadas con codificación de red.

```
7   cfg = ConfigParser.ConfigParser()
8   archivo ="MACaddress.ini"
9   cfg.read(archivo)
10
11  #Importar direcciones MAC del archivo de configuracion
12  MAC_HOST1 = cfg.get('macs','mac_host1')
13  MAC_HOST2 = cfg.get('macs','mac_host2')
14  MAC_HOST3 = cfg.get('macs','mac_host3')
15  MAC_HOST4 = cfg.get('macs','mac_host4')
16  MAC_SWITCH5 = cfg.get('macs','mac_switch5')
17  MAC_SWITCH6 = cfg.get('macs','mac_switch6')
18  MAC_HOST1_U = cfg.get('macs','mac_host1_u')
19  MAC_HOST2_U = cfg.get('macs','mac_host2_u')
20  MAC_HOST3_U = cfg.get('macs','mac_host3_u')
21  MAC_HOST4_U = cfg.get('macs','mac_host4_u')
22  MAC_SWITCH5_U = cfg.get('macs','mac_switch5_u')
23  MAC_SWITCH6_U = cfg.get('macs','mac_switch6_u')
24  MAC_H3XORH4_77 = cfg.get('macs','mac_h3xorh4_77')
25  MAC_H1XORH2_88 = cfg.get('macs','mac_h1xorh2_88')
```

Figura 2.23 Direcciones MAC desde el archivo de configuración

Una vez importadas las librerías necesarias ambas aplicaciones crean un *socket* que permite enviar y recibir tramas desde la interfaz de red de la máquina virtual (ens33). El código para la creación del *socket* se presenta en la Figura 2.24.

```
27  # Creacion del socket para capturar tramas
28  s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0003))
29  s.bind(('ens33', 0))
```

Figura 2.24 Creación del *socket* de la aplicación

El *socket* creado (s) es de tipo *RAW SOCKET*, este tipo de *socket* captura todo tráfico entrante y saliente, y ha sido configurado para capturar tramas Ethernet (0x003) en la interfaz de red a la cual ha sido enlazado (ens33). El *socket* no se enlaza a un puerto específico, por lo que se le asigna el valor 0, que indica que escucha todos los puertos, como se muestra en la línea 29.

En ambas aplicaciones se dispone de varias variables, las cuales están agrupadas de la siguiente forma:

- 6 variables que se utilizarán como vectores para almacenar las tramas procedentes *de* **NC Host1**, **NC Host2**, **NC Host3**, **NC Host4**, **NC Switch5** y **NC Switch6**. Estas variables se presentan en la Figura 2.25.

```
34  # Vectores para tramas de cada VM
35  nchost1_frames = {}
36  nchost2_frames = {}
37  nchost3_frames = {}
38  nchost4_frames = {}
39  ncswitch5_frames = {}
40  ncswitch6_frames = {}
```

Figura 2.25 Variables para almacenar tramas de cada máquina virtual

- 6 variables que servirán para contar las tramas enviadas por cada máquina virtual. Estas variables se presentan en la Figura 2.26.

```
42  #Contador de tramas de cada VM
43  frame_nchost1 = 0
44  frame_nchost2 = 0
45  frame_nchost3 = 0
46  frame_nchost4 = 0
47  frame_ncswitch5 = 0
48  frame_ncswitch6 = 0
```

Figura 2.26 Contadores de tramas

- 4 variables que servirán para contar el número total de tramas recuperadas y perdidas y número de tramas recuperadas para cada terminal. Para la aplicación `CodificadorRedSwitch5.py` se cuentan las tramas que han sido recuperadas y cuyo origen es **NC Host3** o **NC Host4**, mientras que para la aplicación `CodificadorRedSwitch6.py` se cuentan las tramas que han sido recuperadas cuyo origen es **NC Host1** o **NC Host2**. Estas variables se presentan en la Figura 2.27.

```
50  #Contadores para NC:
51  frame_recuperados_total = 0
52  frame_perdidos = 0
53  frames_recuperados_nchost3 = 0
54  frames_recuperados_nchost4 = 0
```

```
50  #Contadores para NC:
51  frame_recuperados_total = 0
52  frame_perdidos = 0
53  frames_recuperados_nchost1 = 0
54  frames_recuperados_nchost2 = 0
```

Figura 2.27 Contadores para codificación de red, `CodificadorRedSwitch5.py` (izquierda) y `CodificadorRedSwitch6.py` (derecha)

- 4 variables que servirán para contar el número de tramas que no se utilizan para codificación de red desde cada terminal, y que se presentan en la Figura 2.28.

```
58  #Contadores para tramas sin NC
59  nchost1_noNC = 0
60  nchost2_noNC = 0
61  nchost3_noNC = 0
62  nchost4_noNC = 0
```

Figura 2.28 Variables para tramas que no son codificadas

- Existe además otras variables usadas para verificar el ingreso a ciertas secciones de código o manejar el reloj del sistema, las cuales se explicarán más adelante.

Ambas aplicaciones funcionan con 3 hilos: El hilo principal de la aplicación encargado de capturar tramas usando el *socket* y procesarlas de acuerdo a su origen; el hilo `visualizador` que muestra el estado de los contadores en pantalla y el hilo `envío_sinNC` que se encarga de enviar tramas sin codificación de red. El código para la creación e inicialización de estos dos últimos hilos se muestra en la Figura 2.29.

```
259  #Hilo para tramas sin NC
260  envio_sinNC = threading.Thread(target=sinNC)
261  envio_sinNC.start()
262
263  #Hilo para visualizacion
264  visualizador = threading.Thread(target=vista)
265  visualizador.start()
```

Figura 2.29 Creación e inicialización de hilos

El hilo `visualizador` ejecuta la función `vista()`la cual despliega el valor actual de las distintas variables en pantalla. El código que despliega la información se presenta en la Figura 2.30 tanto para la aplicación `CodificadorRedSwitch5.py` así como para `CodificadorRedSwitch6.py`.

```
os.system('clear')
print '\nEstadisticas de Paquetes: \n'
print 'frame_nchost1: {0}'.format(frame_nchost1_1)
print 'frame_nchost2: {0}'.format(frame_nchost2_1)
print 'frame_nchost3: {0}'.format(frame_nchost3_1)
print 'frame_nchost4: {0}'.format(frame_nchost4_1)
print 'frame_ncswitch5: {0}'.format(frame_ncswitch5_1)
print 'frame_ncswitch6: {0}'.format(frame_ncswitch6_1)
print 'frames_recuperados_nchost3 : {0}'.format(frames_recuperados_nchost3_1)
print 'frames_recuperados_nchost4 : {0}'.format(frames_recuperados_nchost4_1)
print 'frame_perdidos       : {0}'.format(frame_perdidos_1)
```

```
os.system('clear')
print '\nEstadisticas de Paquetes: \n'
print 'frame_nchost1: {0}'.format(frame_nchost1_1)
print 'frame_nchost2: {0}'.format(frame_nchost2_1)
print 'frame_nchost3: {0}'.format(frame_nchost3_1)
print 'frame_nchost4: {0}'.format(frame_nchost4_1)
print 'frame_ncswitch5: {0}'.format(frame_ncswitch5_1)
print 'frame_ncswitch6: {0}'.format(frame_ncswitch6_1)
print 'frames_recuperados_nchost1 : {0}'.format(frames_recuperados_nchost1_1)
print 'frames_recuperados_nchost2 : {0}'.format(frames_recuperados_nchost2_1)
print 'frame_perdidos       : {0}'.format(frame_perdidos_1)
```

Figura 2.30 Visualización de variables dentro del hilo de visualización, `CodificadorRedSwitch5.py` (izquierda) y `CodificadorRedSwitch6.py` (derecha)

Para facilitar el análisis del funcionamiento de las aplicaciones implementadas el código se analizará de acuerdo a 3 posibles escenarios en donde se envían paquetes tipo ICMP a través de la red y se seguirá paso a paso el proceso de transmisión y recepción de los mensajes.

2.6.4.1. Escenario 1: Transmisión sin pérdidas

En el primer escenario, **NC Host1** y **NC Host2** envían una trama (**p1** o **p2**) a **NC Host3** y **NC Host4** respectivamente. De acuerdo a las reglas del controlador, estas tramas serán enviadas a **s5**, como se muestra en la Figura 2.31.

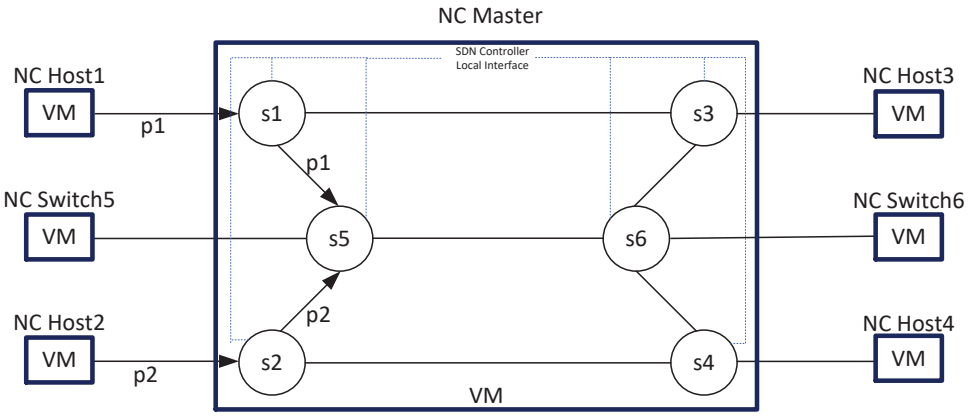
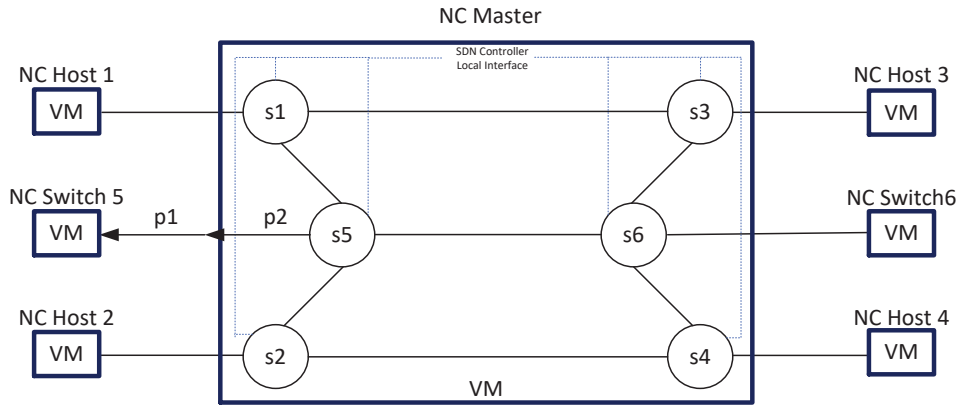


Figura 2.31 Envío de tramas **p1** y **p2** a **s5**

s5 modifica la dirección MAC de destino de las tramas **p1** y **p2**, y los reenvía a **NC Switch5**. Los campos de interés de **p1** y **p2** y el proceso de reenvío a **NC Switch5** se muestran en la Figura 2.32.



Al ingresar a s5				Al salir de s5			
p1		p2		p1		p2	
MAC Destino	MAC Origen	MAC Destino	MAC Origen	MAC Destino	MAC Origen	MAC Destino	MAC Origen
00:00:00:00:00:33	00:00:00:00:00:11	00:00:00:00:00:44	00:00:00:00:00:22	00:00:00:00:00:55	00:00:00:00:00:11	00:00:00:00:00:55	00:00:00:00:00:22

Figura 2.32 Campos de interés de **p1** y **p2** y su reenvío a **NC Switch5**

Al recibir la trama **p1** en el *socket* creado, esta se codifica en un *string* con formato hexadecimal (línea 241), luego se almacenan los campos de la trama Ethernet: MAC de destino (línea 242) y MAC de origen (línea 243), como se muestra en la Figura 2.33.

```
239     # Convertir la trama a string de bytes en Hexadecimal y obtener campos Ethernet
240     packet = packet[0]
241     trama_recibida = packet.encode('hex')
242     destino = trama_recibida[0:12]
243     origen = trama_recibida[12:24]
```

Figura 2.33 Recepción de trama en **NC Switch5**

Posteriormente se procede a verificar el origen de la trama (ver Figura 2.33).

Debido que el *socket* creado captura tanto las tramas entrantes como las salientes, se verifica que la dirección MAC de origen no corresponda a la interfaz de red de **NC Switch5** (00:00:00:00:00:55) ni a la interfaz de red de **NC Master** conectada a **NC Switch5** (00:00:00:00:00:05) (línea 246).

Una vez comprobado esto se verifica el origen de la trama, para lo cual se usa cada una de las direcciones MAC posibles. En este caso la trama proviene de **NC Host1** (línea 249), por lo tanto, se guarda la trama en el vector correspondiente, se actualiza su contador y se cambia la dirección MAC de destino por la dirección original de la trama, que en este caso es la de **NC Host3** (línea 250), para proceder a enviarla de vuelta a **s5** a través del *socket* (línea 253). Este código se muestra en la Figura 2.34.

```
245     #Solo se procesaran tramas que no provengan de manera local o del enlace con el Master
246     if origen not in [MAC_SWITCH6, MAC_SWITCH6_U]:
247
248         #Trama enviada desde el SW3, sentido de Tx ->
249         if origen == MAC_HOST1:
250             destino = MAC_HOST3
251             frame_nchost1 += 1
252             nchost1_frames [frame_nchost1] = destino + trama_recibida[12:]
253             s.send(nchost1_frames [frame_nchost1].decode('hex'))
```

Figura 2.34 Verificación de origen de trama **p1**

Una vez almacenada la trama en el vector correspondiente se procede a verificar si es posible generar tramas codificadas, para lo cual es necesario haber recibido tramas tanto de **NC Host1** como de **NC Host2**.

Un gráfico que ilustra el contenido de los vectores `nchost1_frames` y `nchost2_frames` a manera de un arreglo vertical, y el valor que poseen las variables contadoras `frame_nchost1` `frame_nchost2` hasta este punto de la transmisión para este escenario se muestra en la Figura 2.35.

Figura 2.35 Estado de las variables

Debido a que aún no se ha recibido una trama generada en **NC Host2**, no se puede crear la trama codificada, por lo que el programa debe volver a escuchar el socket para esperar la llegada de otra trama.

Luego de recibir la trama **p1**, la aplicación eventualmente captura la trama **p2**, y el procedimiento anteriormente descrito se repite, actualizando los valores del contador correspondiente y se almacena la trama en su respectivo vector previa la actualización en la dirección MAC de destino. El código de verificación de origen y el grafico del estado de las variables se presenta en la Figura 2.36.

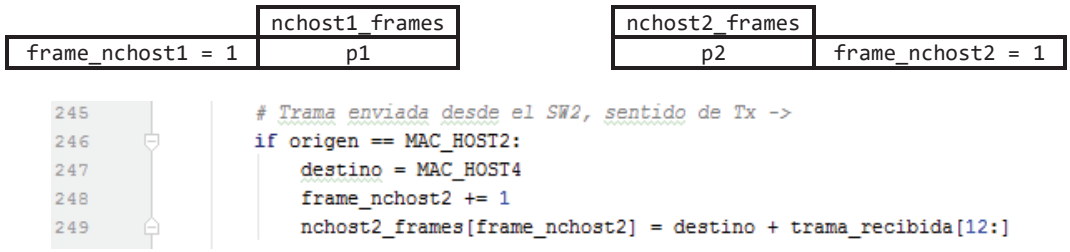


Figura 2.36 Verificación de origen de trama **p2** y estado de variables

Con las dos tramas recibidas y almacenadas en sus respectivos vectores, se cumple la condición de la línea 287 de la Figura 2.37, luego de lo cual se crea la trama codificada realizando una operación `xor` entre **p1** y **p2** (línea 292) realizada de la siguiente manera:

- La trama codificada inicia con el string `0000000000` debido a que al realizar las distintas operaciones descritas a continuación se pierden los ceros iniciales que posee la trama codificada.
- El resto del string de la trama codificada se obtiene a partir de la suma binaria de las tramas convertidas a hexadecimal utilizando una conversión a entero de base 16 (`int` con argumento 16) y el operador `^`. Una vez realizada la suma binaria se procede a convertir el resultado a una serie de dígitos que pueden ser traducidos a `string` utilizando la función `format` con el código `02x`.

Una vez generada la nueva trama codificada se la almacena en el vector de tramas de **NC Switch5** modificando su dirección MAC origen por la dirección MAC especificada en los archivos de configuración (00:00:00:00:00:88).

Una vez realizado este procedimiento se envían las 3 tramas a la red a través del método `send()` del *socket* (líneas 397, 398 y 399). El código código de la aplicación y el estado de las variables al momento de ejecutar esta sección se presentan en la Figura 2.37.

frame_nchost1 = 1	nchost1_frames	nchost2_frames		ncswitch5_frames	
	p1	p2	frame_nchost2 = 1	p1 xor p2	frame_ncswitch5 = 1

```
286         #Verificar si se pueden crear tramas codificadas
287         if (frame_nchost1 > 0 or frame_nchost2 > 0) and origen in [MAC_HOST1, MAC_HOST2]:
288
289             if frame_nchost1 == frame_nchost2 and frame_nchost1 > frame_ncswitch5 and frame_nchost2 > frame_ncswitch5:
290
291                 frame_ncswitch5 += 1
292                 trama_codificada = '0000000000' + str(format(int(nchost1_frames[frame_nchost1], 16) ^
293                                                              int(nchost2_frames[frame_nchost2], 16), '02x'))
294                 ncswitch5_frames[frame_ncswitch5] = trama_codificada[0:12] + MAC_H1XORH2_88 + trama_codificada[24:]
295
296                 #Enviar Tramas
297                 s.send(nchost1_frames[frame_nchost1].decode('hex'))
298                 s.send(nchost2_frames[frame_nchost2].decode('hex'))
299                 s.send(ncswitch5_frames[frame_ncswitch5].decode('hex'))
```

Figura 2.37 Codificación, envío de tramas y estado de las variables

Las 3 tramas se envían a **s5** y de acuerdo con las reglas instaladas en el *switch,* estas se reenvían a **s1**, **s2** y **s6,** como se muestra en la Figura 2.38.

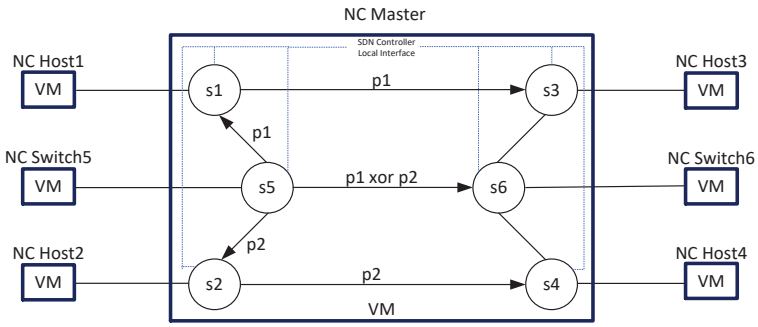
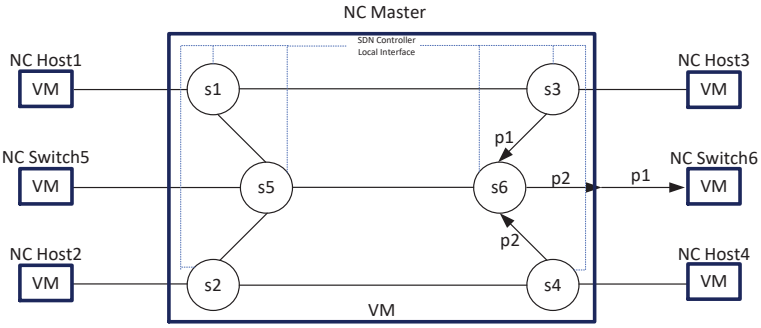


Figura 2.38 Transmisión de tramas desde **s5**

Debido a los retardos programados dentro de Mininet en cada uno de los enlaces, las tramas **p1** y **p2** llegarán a **s3** y **s4** y serán reenviados a **s6** antes que la trama codificada **p1 xor p2** llegue a **s6**. Cuando **p1** y **p2** lleguen a **s6** se modificará la dirección MAC destino por la de **NC Switch6** y serán reenviadas de acuerdo a las reglas, como se muestra en la Figura 2.39.



Al ingresar a s6				Al salir de s6			
p1		p2		p1		p2	
MAC Destino	MAC Origen	MAC Destino	MAC Origen	MAC Destino	MAC Origen	MAC Destino	MAC Origen
00:00:00:00:00:33	00:00:00:00:00:11	00:00:00:00:00:44	00:00:00:00:00:22	00:00:00:00:00:66	00:00:00:00:00:11	00:00:00:00:00:66	00:00:00:00:00:22

Figura 2.39 Campos de interés de **p1** y **p2 y** su reenvío a **NC Switch6**

Una vez recibido una trama se la procesa de la misma manera que el caso anterior, convirtiéndola a un *string* y obteniendo los campos de interés.

Después se verifica el origen de las tramas recibidas, en caso de proceder de **NC Host1** (00:00:00:00:00:11) o **NC Host2** (00:00:00:00:00:22) únicamente es necesario actualizar los contadores, almacenar las tramas en sus respectivos vectores y reenviarlas usando el *socket*, como se muestra en la Figura 2.40.

```
245     #Solo se procesaran tramas que no provengan de manera local o del enlace con el Master
246     if origen not in [MAC_SWITCH6, MAC_SWITCH6_U]:
247
248         #Trama enviada desde el SW3, sentido de Tx ->
249         if origen == MAC_HOST1:
250             destino = MAC_HOST3
251             frame_nchost1 += 1
252             nchost1_frames [frame_nchost1] = destino + trama_recibida[12:]
253             s.send(nchost1_frames [frame_nchost1].decode('hex'))
254
255         # Trama enviada desde el SW4, sentido de Tx ->
256         if origen == MAC_HOST2:
257             destino = MAC_HOST4
258             frame_nchost2 += 1
259             nchost2_frames [frame_nchost2] = destino + trama_recibida[12:]
260             s.send(nchost2_frames [frame_nchost2].decode('hex'))
```

Figura 2.40 Verificación de origen de tramas desde **NC Host1** y **NC Host2**

Después de haberse enviado tanto **p1** y **p2** a **s6**, se los reenvían a **NC Host3** y **NC Host4** respectivamente; posteriormente la trama codificada llega a **s6** y es reenviada a **NC Switch6** para su procesamiento. Este proceso se muestra en la Figura 2.41.

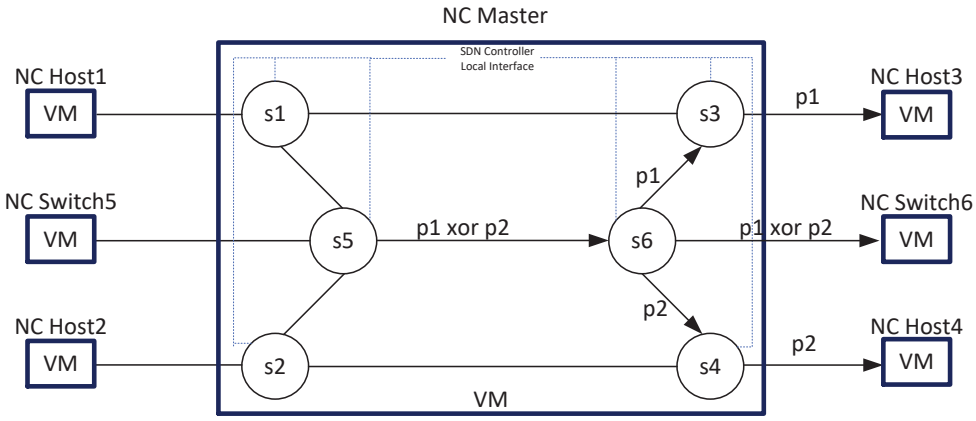


Figura 2.41 Reenvío de **p1**, **p2** y trama codificada

Una vez que la trama codificada llega a **NC Switch6** se la procesa ingresando a la condición de verificación de origen, se actualizará el respectivo contador y se

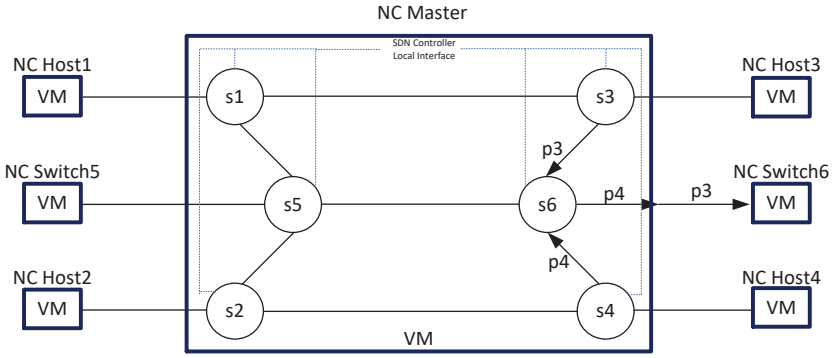
almacenará la trama en el vector para tramas de **NC Switch5** editando la dirección MAC destino por la dirección MAC especificada en el archivo de configuración (00:00:00:00:00:77). Una vez almacenada la trama se verifica si es necesario realizar recuperación. En este caso no es necesario ya que se han recibido previamente las tramas **p1** y **p2**, y por lo tanto no requiere decodificación. Por lo que se vuelve a esperar la llegada de una nueva trama. Este código y el estado de las variables se muestran en la Figura 2.42.

nchost1_frames		nchost2_frames		ncswitch5_frames	
frame_nchost1 = 1	p1	p2	frame_nchost2 = 1	p1 xor p2	frame_ncswitch5 = 1

```
367         if origen == MAC_H1XORH2_88:
368
369             espera = False
370             reloj = clock()
371             tiempo = 0
372
373             destino = MAC_H3XORH4_77
374             frame_ncswitch5 += 1
375             ncswitch5_frames [frame_ncswitch5] = destino + trama_recibida[12:]
376
377             #Si se han recibido previamente tramas del Host 1 y 2
378
379             if frame_nchost1 >= frame_ncswitch5 and frame_nchost2 >= frame_ncswitch5:
380
381                 espera = False
382                 reloj = clock()
383                 tiempo = 0
```

Figura 2.42 Procesamiento de trama codificada sin recuperación

Una vez que **NC Host3** y **NC Host4** han recibido **p1** y **p2** envían sus mensajes de respuesta **p3** y **p4,** y de acuerdo a las reglas del controlador las tramas enviadas por ambos terminales viajan hacia **s6** para ser reenviadas a **NC Switch6** editando su dirección MAC de origen. Este proceso y los campos de interés de **p3** y **p4** se muestran en la Figura 2.43.



Al ingresar a s6				Al salir de s6			
p3		p4		p3		p4	
MAC Destino	MAC Origen	MAC Destino	MAC Origen	MAC Destino	MAC Origen	MAC Destino	MAC Origen
00:00:00:00:00:11	00:00:00:00:00:33	00:00:00:00:00:22	00:00:00:00:00:44	00:00:00:00:00:66	00:00:00:00:00:33	00:00:00:00:00:66	00:00:00:00:00:44

Figura 2.43 Campos de interés de **p3** y **p4** y su reenvío a **NC Switch6**

Al recibir la trama en **NC Switch6** se procesa de la misma manera en que se procesa en **NC Switch5**, convirtiendo la trama a un *string* y obteniendo los campos de interés.

Posteriormente se procede a verificar el origen de la trama recibida, para el caso de **p3** y **p4** el código se presenta en la Figura 2.44 en el cual se actualizan las variables correspondientes y se almacenan las tramas en los vectores editando su dirección MAC de destino.

```
262         # Trama enviada desde el SW3, sentido de Tx <-
263         if origen == MAC_HOST3:
264             destino = MAC_HOST1
265             frame_nchost3 += 1
266             nchost3_frames [frame_nchost3] = destino + trama_recibida[12:]
267
268         # Trama enviada desde el SW4, sentido de Tx <-
269         if origen == MAC_HOST4:
270             destino = MAC_HOST2
271             frame_nchost4 += 1
272             nchost4_frames [frame_nchost4] = destino + trama_recibida[12:]
```

Figura 2.44 Verificación de origen para **p3** y **p4**

Con las dos tramas recibidas y almacenadas en sus respectivos vectores, la generación de la trama codificada se realiza de la misma manera que para **p1** y **p2** considerando únicamente las variables para contadores y vectores correspondientes a los terminales de interés (**NC Host3** y **NC Host4**).

Este código de la aplicación y el estado de las variables se presentan en la Figura 2.45.

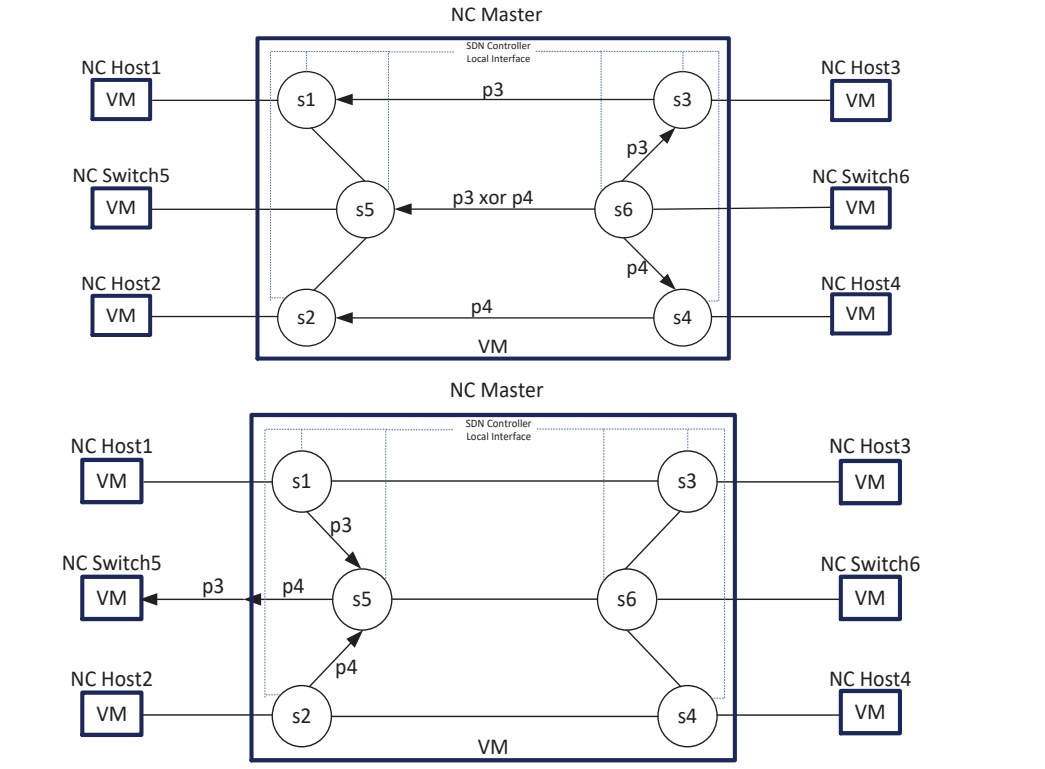
	nchost3_frames	nchost4_frames		ncswitch6_frames	
frame_nchost3 = 1	p3	p4	frame_nchost4 = 1	p3 xor p4	frame_ncswitch6 = 1

```
294         #Verificar si se pueden crear tramas codificadas
295         if (frame_nchost3 > 0 or frame_nchost4 > 0) and origen in [MAC_HOST3, MAC_HOST4]:
296
297             if frame_nchost3 == frame_nchost4 and frame_nchost3 > frame_ncswitch6 and frame_nchost4 > frame_ncswitch6:
298
299                 frame_ncswitch6 += 1
300                 trama_codificada = '0000000000' + str(format(int(nchost3_frames [frame_nchost3], 16) ^
301                                                              int(nchost4_frames [frame_nchost4], 16), '02x'))
302                 ncswitch6_frames [frame_ncswitch6] = MAC_H1XORH2_88 + trama_codificada[12:]
303
304                 #Enviar Tramas
305                 s.send(nchost3_frames [frame_nchost3].decode('hex'))
306                 s.send(nchost4_frames [frame_nchost4].decode('hex'))
307                 s.send(ncswitch6_frames [frame_ncswitch6].decode('hex'))
```

Figura 2.45 Codificación, envío de tramas y estado de las variables

A partir de este punto, la transmisión de **p3** y **p4,** así como de su trama codificada (**p3 xor p4**) se realiza en sentido inverso a la transmisión de **p1**, **p2** y **p1 xor p2**.

Esto se muestra en la Figura 2.46 donde las tres tramas son reenviadas hasta llegar a **NC Switch5**.



Al ingresar a s5				Al salir de s5			
p3		p4		p3		p4	
MAC Destino	MAC Origen	MAC Destino	MAC Origen	MAC Destino	MAC Origen	MAC Destino	MAC Origen
00:00:00:00:00:11	00:00:00:00:00:33	00:00:00:00:00:22	00:00:00:00:00:44	00:00:00:00:00:55	00:00:00:00:00:33	00:00:00:00:00:66	00:00:00:00:00:44

Figura 2.46 Reenvío de **p3**, **p4** y trama codificada (arriba). Campos de interés de **p3** y **p4** **y** su reenvío a **NC Switch5** (abajo)

Una vez recibida una trama, ya sea **p3** o **p4** en **NC Switch5** se la procesa de la misma manera que el caso anterior, convirtiéndola a un *string* y obteniendo los campos de internes como se muestra en el código de la Figura 2.47.

```
251
252     # Trama enviada desde el SW1, sentido de Tx <-
253     if origen == MAC_HOST3:
254         destino = MAC_HOST1
255         frame_nchost3 += 1
256         nchost3_frames[frame_nchost3] = destino + trama_recibida[12:]
257         s.send(nchost3_frames[frame_nchost3].decode('hex'))
258
259     # Trama enviada desde el SW2, sentido de Tx <-
260     if origen == MAC_HOST4:
261         destino = MAC_HOST2
262         frame_nchost4 += 1
263         nchost4_frames[frame_nchost4] = destino + trama_recibida[12:]
264         s.send(nchost4_frames[frame_nchost4].decode('hex'))
```

Figura 2.47 Verificación de origen de tramas desde **NC Host3** y **NC Host4**

p3 y **p4** se reenvían a **NC Host1** y **NC Host2** respectivamente, mientras que la trama codificada llega a **s5** y es reenviada a **NC Switch5** para su procesamiento como se muestra en la Figura 2.48.

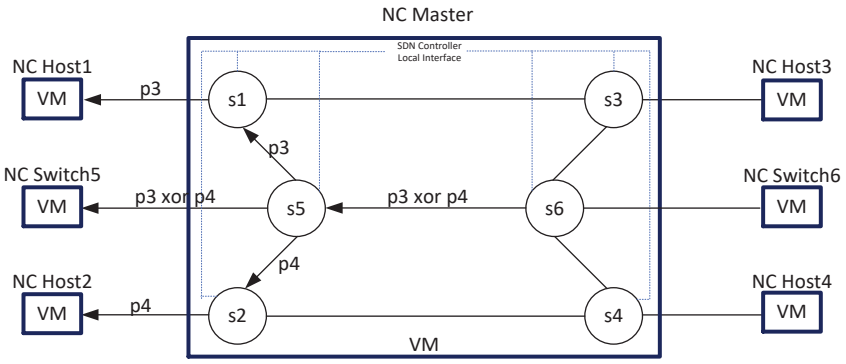


Figura 2.48 Reenvío de **p3**, **p4** y trama codificada

Se procede a verificar si es necesario recuperar tramas de la misma manera que para **p1** y **p2** como se muestra en el código de la Figura 2.49. En este caso tampoco es necesario recuperar tramas.

```
nchost3_frames                nchost4_frames                ncswitch6_frames
frame_nchost3 = 1 | p3        p4 | frame_nchost4 = 1        p3 xor p4 | frame_ncswitch6 = 1

356         #Tratamiento de trama NC
357         if origen == MAC_H3XORH4_77:
358
359             destino = MAC_H1XORH2_88
360             frame_ncswitch6 += 1
361             ncswitch6_frames[frame_ncswitch6] = destino + trama_recibida[12:]
362
363             #Si se han recibido previamente tramas del Host 3 y 4
364             if frame_nchost3 >= frame_ncswitch6 and frame_nchost4 >= frame_ncswitch6:
365
366                 espera = False
367                 reloj = clock()
368                 tiempo = 0
```

Figura 2.49 Procesamiento de trama codificada sin recuperación

2.6.4.2. Escenario 2: Transmisión con pérdidas

En el segundo escenario, **NC Host1** envía una trama **p1** a **NC Host3** mientras que **NC Host2** envía una trama **p2** a **NC Host4**.

En este escenario se analiza la pérdida de tramas en los enlaces entre **s1** y **s3** y entre **s2** y **s4**.

La trasmisión se da de la misma manera que en el escenario anterior, hasta el punto de obtener la trama codificada y enviar **p1**, **p2** y **p1 xor p2** por la red.

La trama **p2** se pierde debido a una falla producida en el enlace entre **s2** y **s4**, como se muestra en la Figura 2.50 donde el paquete **p2** se pierde debido a un error en el enlace.

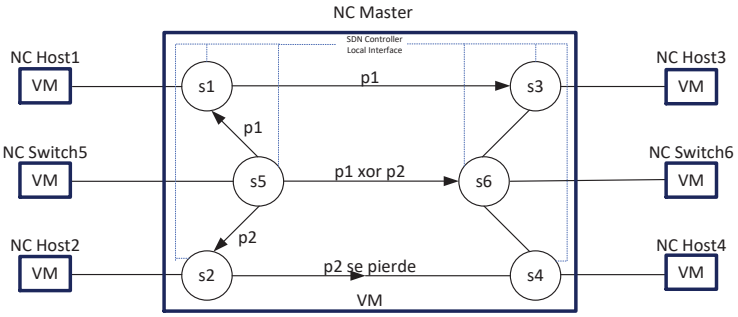
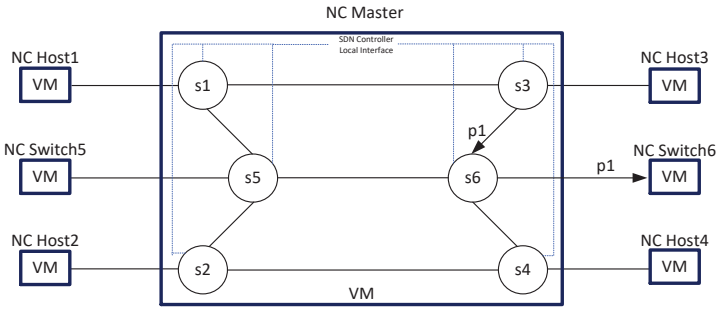


Figura 2.50 Reenvío de **p1**, **p2** y trama codificada. Pérdida de **p2**

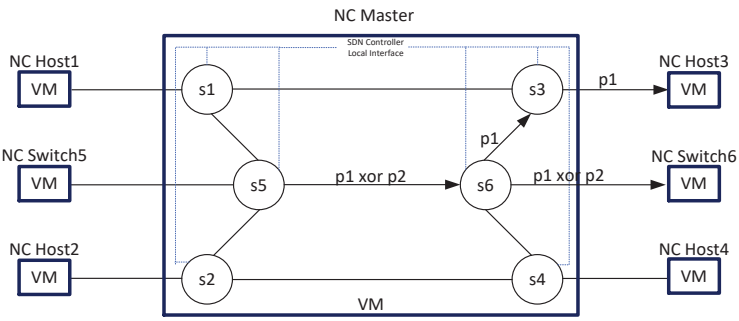
Debido a la pérdida en el enlace entre **s2** y **s4**, **p2** no llega a **s4** mientras que debido a los retardos programados, **p1** llegará a **s3** y será reenviado a **s6** antes que la trama codificada llegue a **s6**. Cuando **p1** llegue a **s6** será reenviado a **NC Switch6** de acuerdo a las reglas instaladas como se muestra en la Figura 2.51.



Al ingresar a s6		Al salir de s6	
p1		p1	
MAC Destino	MAC Origen	MAC Destino	MAC Origen
00:00:00:00:00:33	00:00:00:00:00:11	00:00:00:00:00:66	00:00:00:00:00:11

Figura 2.51 Campos de interés de **p1** y su reenvío a **NC Switch6**

Una vez procesada **p1** de la misma manera que en el escenario anterior, es reenviada para entregarla a **NC Host3**, como se muestra en la Figura 2.52.



Al ingresar a s6		Al salir de s6	
MAC Destino	MAC Origen	MAC Destino	MAC Origen
00:00:00:00:00:77	00:00:00:00:00:88	00:00:00:00:00:66	00:00:00:00:00:88

Figura 2.52 Campos de interés de **p1xor p2** y su reenvío a **NC Switch6**

Al llegar la trama codificada, esta se procesa de la misma manera que en el escenario 1 presentado anteriormente. En este escenario el estado de las variables es diferente, y el programa ejecuta la acción de recuperación de acuerdo a la condición de la línea 390 del código (ver Figura 2.53), en donde se verifica que ha llegado una trama desde **NC Host1**, y la trama codificada, sin embargo, la trama desde **NC Host2** no ha llegado y debe ser recuperada. Con el fin de recuperar la trama perdida se realiza una operación XOR entre la trama codificada y **p1** (línea 496). Una vez obtenida la trama original **p2** se actualizan los contadores de **NC Host2** y se almacena la trama en el vector correspondiente, igualando así las variables y permitiendo procesar nuevas tramas que lleguen a **NC Switch6**. Una vez realizado este procedimiento se reenvía **p2** usando el método `send()` del *socket* (línea 403).

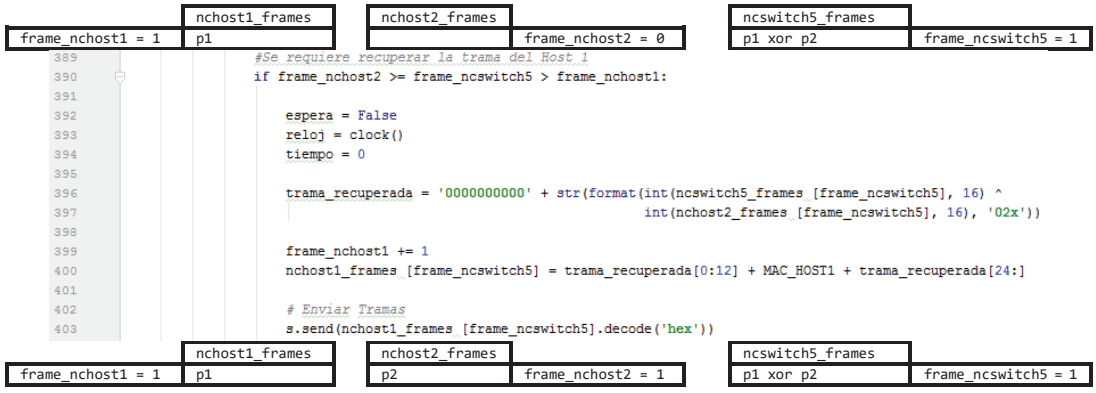


Figura 2.53 Variables previo a recuperación, código de recuperación y variables luego de la recuperación

Una vez recuperado **p2** se lo reenvía a **s6**, para posteriormente ser reenviado a **s4** y entregado en **NC Host4** como se muestra en la Figura 2.54.

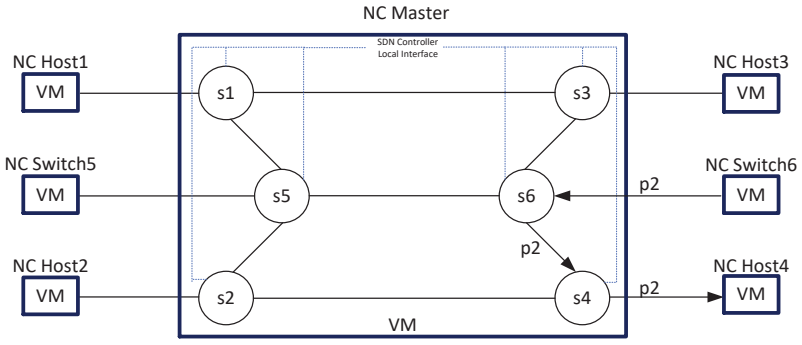


Figura 2.54 Reenvío de **p2** a **NC Host4**

La trasmisión de las tramas de respuesta **p3** y **p4** se da de la misma manera que en el escenario anterior. La trama **p3** se pierde como se muestra en la Figura 2.55.

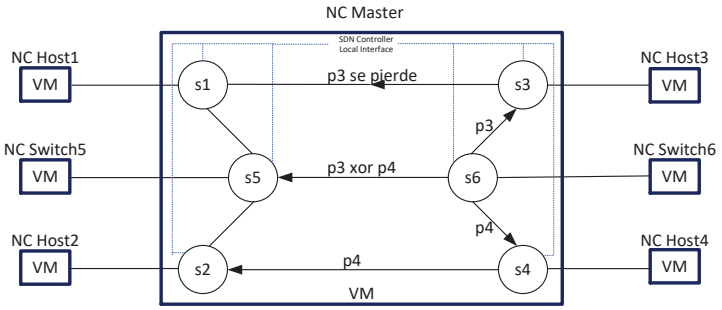
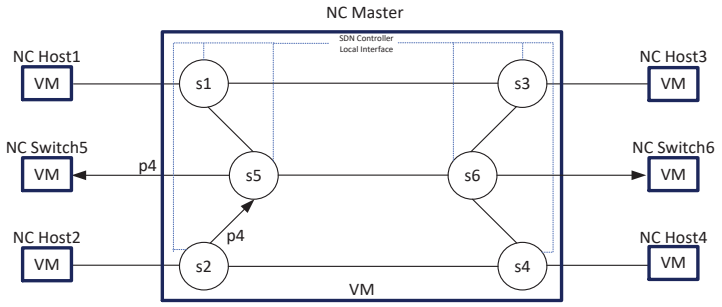


Figura 2.55 Reenvío de **p3**, **p4** y trama codificada

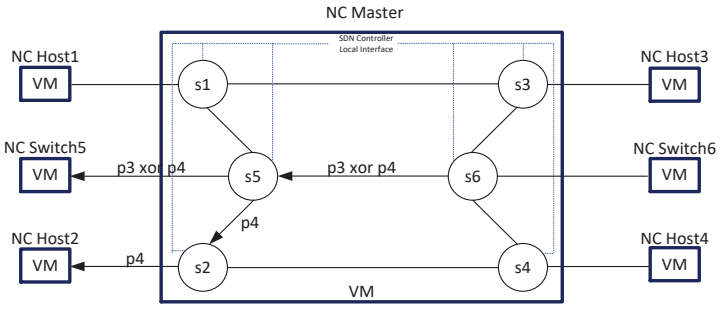
Al perderse **p3** solo **p4** llegará a **NC Switch5**, como se muestra en la Figura 2.56.



Al ingresar a s5		Al salir de s5	
p4		p4	
MAC Destino	MAC Origen	MAC Destino	MAC Origen
00:00:00:00:00:22	00:00:00:00:00:44	00:00:00:00:00:55	00:00:00:00:00:44

Figura 2.56 Campos de interés de **p4** y su reenvío a **NC Switch5**

Una vez procesada **p4**, es reenviada para entregarla a **NC Host2** y se espera la llegada de una nueva trama en **NC Switch5**, que para este escenario será la trama codificada ya que **p3** se ha perdido como se muestra en la Figura 2.57.



Al ingresar a s5		Al salir de s5	
MAC Destino	MAC Origen	MAC Destino	MAC Origen
00:00:00:00:00:88	00:00:00:00:00:77	00:00:00:00:00:66	00:00:00:00:00:77

Figura 2.57 Campos de interés de **p3 xor p4** y su reenvío a **NC Switch5**

La trama codificada se procesa de la misma manera que en la trasmisión anterior para recuperar **p3**, como se muestra en la Figura 2.58 y se lo entrega a **NC Host1** como se muestra en la Figura 2.59.

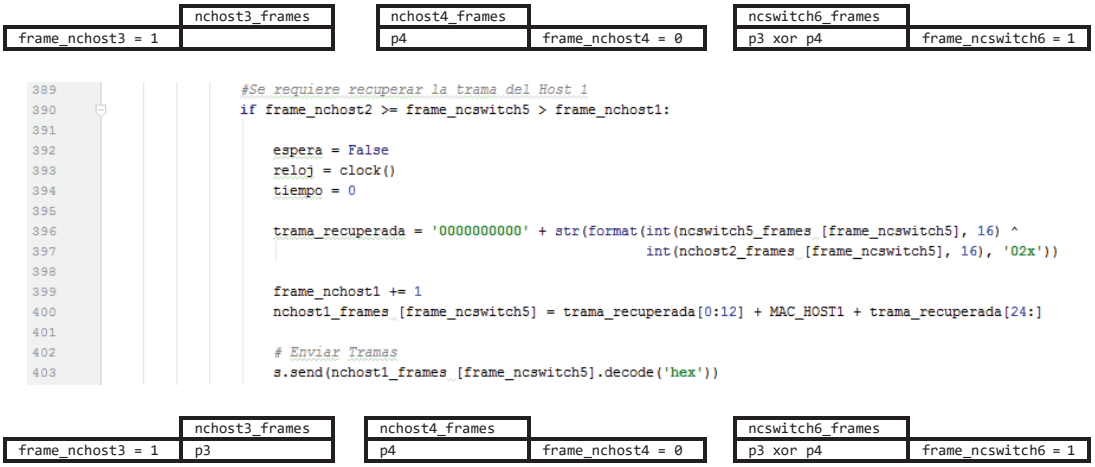


Figura 2.58 Variables previo a recuperación, código de recuperación y variables luego de la recuperación

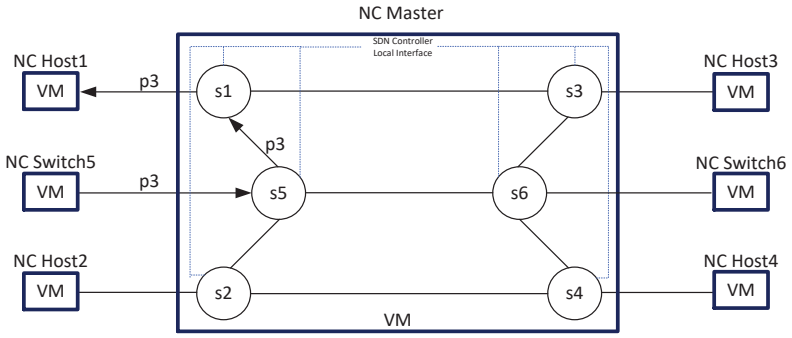


Figura 2.59 Reenvío de **p3** a **NC Host1**

El escenario presentado explica el código generado para recuperar las tramas que han sido trasmitidas desde **NC Host2** y tramas que han sido transmitidas desde NC **Host3**.

Sin embargo, el proceso para recuperar tramas que han sido transmitidas desde **NC Host1** y **NC Host4** es el mismo, con variaciones únicamente en las variables utilizadas.

Para el proceso de recuperación de tramas de **NC Host1** el código y el estado de las variables antes y después del proceso de recuperación en **NC Switch6** se muestran en la Figura 2.60.

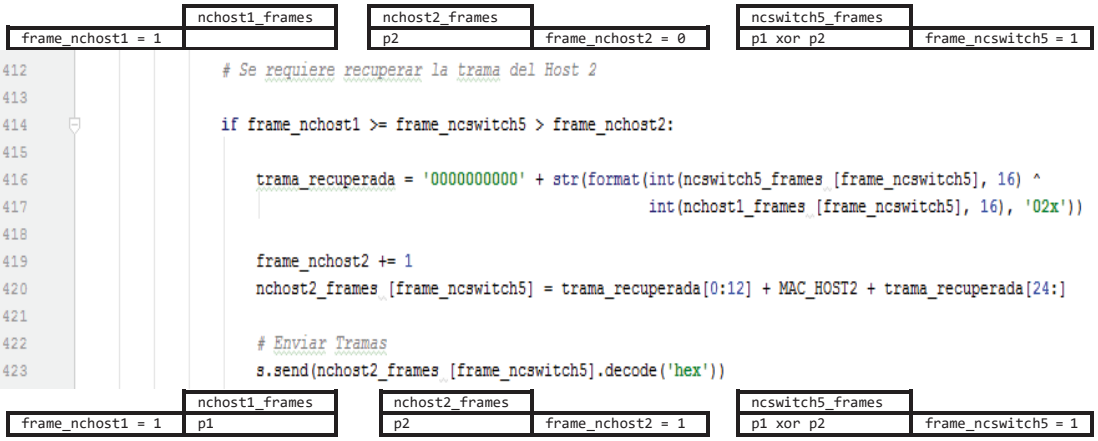


Figura 2.60 Variables previo a recuperación, código de recuperación y variables luego de la recuperación

Para el proceso de recuperación de tramas de **NC Host4** el código y el estado de las variables antes y después del proceso de recuperación en **NC Switch5** se muestran en la Figura 2.61.

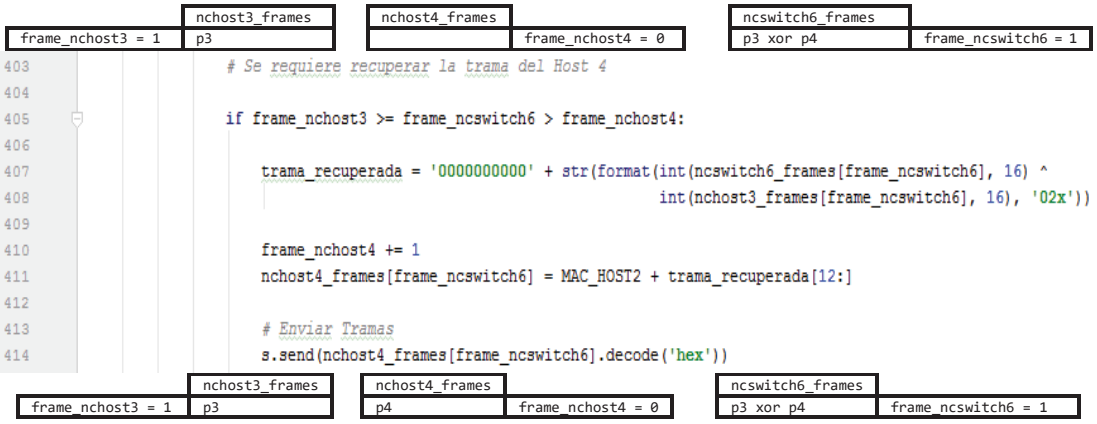


Figura 2.61 Variables previo a recuperación, código de recuperación y variables luego de la recuperación

2.6.4.3. Escenario 3: Transmisión sin utilizar codificación de red

En el tercer escenario, **NC Host1** envía una trama **p1** a **NC Host3** y que responde con **p3**.

De acuerdo a las reglas del controlador **p1** viaja hacia **s5** para ser reenviado a **NC Switch5** editando su dirección MAC de origen como se analizó en los escenarios anteriores.

Una vez que **p1** ha sido procesado, ha sido almacenado en el vector y los contadores fueron actualizados, el código del programa procede a verificar si se puede crear una trama codificada, debido a que no se recibe una trama desde **NC Host2** para crear dicha trama, el hilo principal del programa entra a esperar. El estado de las variables para este caso se muestra en la Figura 2.62.

Figura 2.62 Estado de las variables

Como se explicó en la Figura 2.28, existe un hilo llamado `envío_sinNC()` que ejecuta el método `sinNC()`. Este método está programado para activarse cada vez que se recibe y procesa una trama, y se desactiva cada vez que se genera una trama codificada o se realiza el proceso de recuperación. Este control se realiza con un bucle `while` controlado por una variable espera, que cambia su valor de `True` a `False` para activar o desactivar las acciones del método `sinNC()`. Este método utiliza el reloj del sistema para contar 1 segundo cada vez que es activada la variable espera. Al desactivarse esta variable el contador del reloj vuelve a cero.

Si ha transcurrido 1 segundo y la variable espera no ha sido desactivada quiere decir que no se ha generado una trama codificada ni se ha realizado recuperación, por lo tanto, existe una trama que debe ser enviada sin usar codificación de red. En este caso se requiere enviar **p1** sin codificación de red, por lo cual el método `sinNC()` espera un segundo y posteriormente reemplaza la dirección de origen por aquella especificada en el archivo de configuración para tramas sin codificación. Actualiza los contadores necesarios con el fin de igualar todos los valores de la red y envía esta trama a **s5** para ser entregada en **NC Host3**. El código para este procedimiento se muestra en la Figura 2.63.

```
98      while espera:
99          tiempo = 1
100         if clock() - reloj > 1 and espera:
101             if frame_nchost1 > frame_nchost2 == frame_ncswitch5 and espera and clock() - reloj > 1:
102
103                 trama_sw1 = nchost1_frames[frame_nchost1]
104                 trama_sw1 = trama_sw1[0:12] + MAC_HOST1_U + trama_sw1[24:]
105                 s.send(trama_sw1.decode('hex'))
106
107                 frame_nchost2 += 1
108                 frame_ncswitch5 += 1
109                 noNCframes += 1
110                 nchost1_noNC += 1
111                 if frame_nchost1 == frame_nchost2:
112                     espera = False
113                     tiempo = 0
```

Figura 2.63 Envío de **p1** desde **NC Switch5**

El viaje de **p1** a través de la red se muestra en la Figura 2.64 donde se muestra en primer lugar el reenvío de **p1** hasta **NC Switch5**; en segundo lugar, el reenvío de **p1** una vez que ha transcurrido un segundo de espera hasta su recepción en **NC Switch6** y en tercer lugar se muestra su reenvío desde **NC Switch6** hasta su destino final en **NC Host3**.

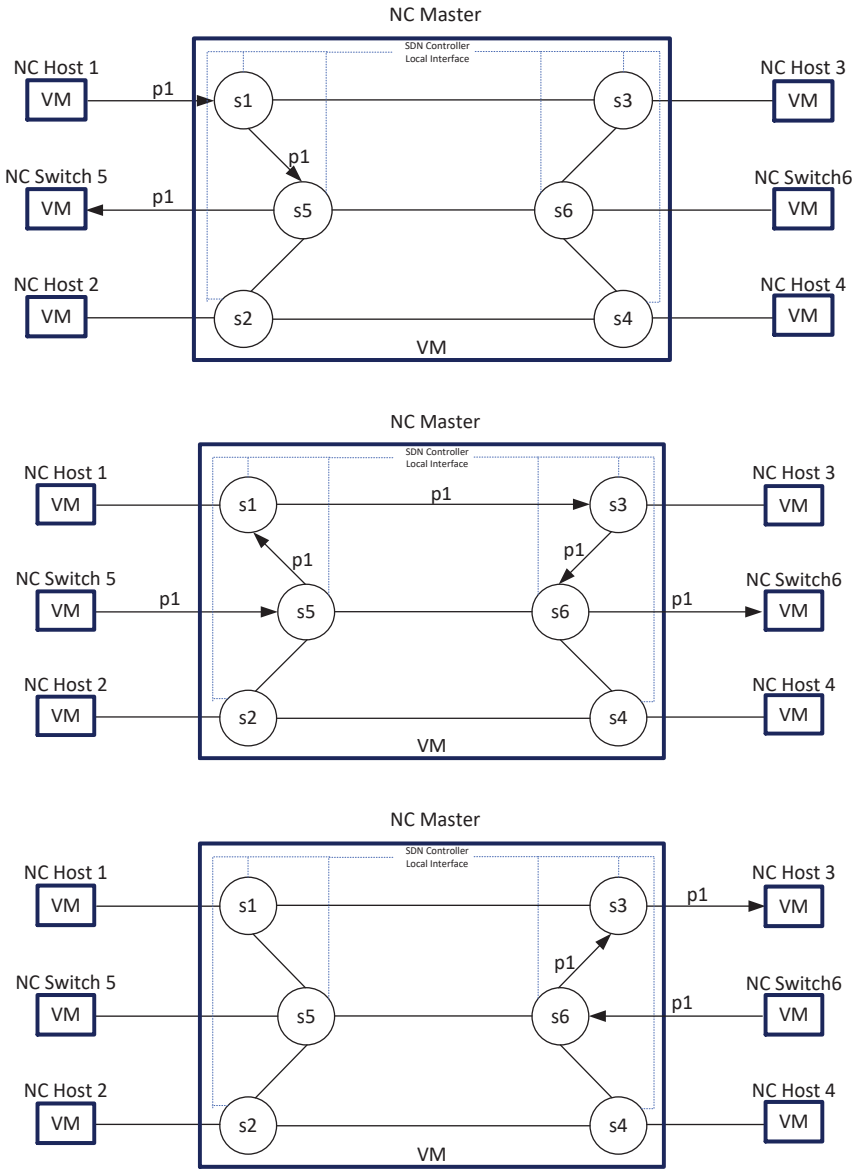


Figura 2.64 Proceso de reenvío de **p1** desde **NC Host1** a **NC Host3**

Al llegar a **NC Switch6**, **p1** se procesa de la misma manera que el resto de tramas, actualizando además su dirección MAC de origen por la dirección MAC de **NC Host1** como se muestra en la Figura 2.65.

```
274         # Trama unica enviada desde el SW3, sentido de Tx ->
275         if origen == MAC_HOST1_U:
276             origen = MAC_HOST1
277             destino = MAC_HOST3
278             frame_nchost1 += 1
279             frame_nchost2 += 1
280             frame_ncswitch5 += 1
281             nchost1_frames [frame_nchost1] = destino + origen + trama_recibida[24:]
282             s.send(nchost1_frames [frame_nchost1].decode('hex'))
```

Figura 2.65 Recepción de **p1** en **NC Switch6**

Una vez que **NC Host3** recibe **p1** genera la respuesta **p3** y la envía a **NC Host1**. De la misma manera que con **p1**, el hilo secundario se encarga de enviar esta trama sin codificación de red en base al código de la Figura 2.66.

```
95  while True:
96
97      reloj = clock()
98
99      while espera:
100         tiempo = 1
101         if clock() - reloj > 1 and espera:
102             if frame_nchost3 > frame_nchost4 == frame_ncswitch6 and espera and clock() - reloj > 1:
103
104                 trama_sw3 = nchost3_frames [frame_nchost3]
105                 trama_sw3 = trama_sw3[0:12] + MAC_HOST3_U + trama_sw3[24:]
106                 s.send(trama_sw3.decode('hex'))
107
108                 frame_nchost4 += 1
109                 frame_ncswitch6 += 1
110                 noNCframes += 1
111                 sw3_noNC += 1
112                 if frame_nchost3 == frame_nchost4:
113                     espera = False
114                     tiempo = 0
```

Figura 2.66 Envío de **p3** desde **NC Switch6**

Al llegar a **NC Switch5**, **p3** se procesa de la misma manera que el resto de tramas, actualizando además su dirección MAC de origen por la dirección MAC de **NC Host3** como se muestra en la Figura 2.67.

```
266         # Trama unica enviada desde el SW1, sentido de Tx <-
267         if origen == MAC_HOST3_U:
268             origen = MAC_HOST3
269             destino = MAC_HOST1
270             frame_nchost3 += 1
271             frame_nchost4 += 1
272             frame_ncswitch6 += 1
273             nchost3_frames[frame_nchost3] = destino + origen + trama_recibida[24:]
274             s.send(nchost3_frames[frame_nchost3].decode('hex'))
```

Figura 2.67 Recepción de **p3** en **NC Switch5**

EL proceso de reenvío de **p3** es similar al mostrado para **p1** en la Figura 2.64 en sentido inverso.

En caso del envío de **p2** desde **NC Host2** para **NC Host4** sin existir **p1** se realiza un procedimiento similar con las variables correspondientes. El código a ejecutar para **p2** en **NC Switch5** se muestra en la Figura 2.68 y el código a ejecutar para su respuesta en **NC Switch6** se muestra en la Figura 2.69.

```
114         if frame_nchost2 > frame_nchost1 == frame_ncswitch5 and espera and clock() - reloj > x:
115
116             trama_sw2 = nchost2_frames[frame_nchost2]
117             trama_sw2 = trama_sw2[0:12] + MAC_HOST2_U + trama_sw2[24:]
118             s.send(trama_sw2.decode('hex'))
119
120             frame_nchost1 += 1
121             frame_ncswitch5 += 1
122             noNCframes += 1
123             nchost2_noNC += 1
124             if frame_nchost2 == frame_nchost1:
125                 espera = False
126                 tiempo = 0
```

Figura 2.68 Envío de **p2** desde **NC Switch5**

```
284     # Trama unica enviada desde el SW4, sentido de Tx ->
285     if origen == MAC_HOST2_U:
286         origen = MAC_HOST2
287         destino = MAC_HOST4
288         frame_nchost1 += 1
289         frame_nchost2 += 1
290         frame_ncswitch5 += 1
291         nchost2_frames [frame_nchost2] = destino + origen + trama_recibida[24:]
292         s.send(nchost2_frames [frame_nchost2].decode('hex'))
```

Figura 2.69 Recepción de **p2** en **NC Switch6**

En caso del envío de **p4** desde **NC Host4** para **NC Host2** sin existir **p3** se realiza un procedimiento similar con las variables correspondientes. El código a ejecutar para **p4** en **NC Switch6** se muestra en la Figura 2.70 y el código a ejecutar en **NC Switch5** se muestra en la Figura 2.71.

```
115         if frame_nchost4 > frame_nchost3 == frame_ncswitch6 and espera and clock() - reloj > x:
116
117             trama_sw4 = nchost4_frames [frame_nchost4]
118             trama_sw4 = trama_sw4[0:12] + MAC_HOST4_U + trama_sw4[24:]
119             s.send(trama_sw4.decode('hex'))
120
121             frame_nchost3 += 1
122             frame_ncswitch6 += 1
123             noNCframes += 1
124             sw4_noNC += 1
125             if frame_nchost4 == frame_nchost3:
126                 espera = False
127                 tiempo = 0
```

Figura 2.70 Envío de **p4** desde **NC Switch6**

```
276     # Trama unica enviada desde el SW2, sentido de Tx <-
277     if origen == MAC_HOST4_U:
278         origen = MAC_HOST4
279         destino = MAC_HOST2
280         frame_nchost3 += 1
281         frame_nchost4 += 1
282         frame_ncswitch6 += 1
283         nchost4_frames[frame_nchost4] = destino + origen + trama_recibida[24:]
284         s.send(nchost4_frames[frame_nchost4].decode('hex'))
```

Figura 2.71 Recepción de **p4** en **NC Switch5**

El código completo de estas aplicaciones se encuentra en los Anexos V y VI.

2.7. PRUEBAS DE FUNCIONAMIENTO

En esta sección se muestra un resumen de las pruebas de funcionamiento realizadas en este proyecto. Se presenta la ejecución de las máquinas virtuales, su

conexión con la red con topología tipo *butterfly*, y pruebas de funcionamiento sin codificación de red y con codificación de red para cada uno de los escenarios descritos.

Se utiliza el comando `ping` para generar tráfico ICMP, la aplicación `at` para sincronizar el envío de información y el capturador de tráfico Wireshark para verificar el funcionamiento de las aplicaciones en el escenario.

2.7.1. EJECUCIÓN Y CONEXIÓN DE LAS MÁQUINAS VIRTUALES

Las 7 máquinas virtuales requeridas para el proyecto se ejecutan de forma simultánea dentro de un equipo físico, cada una de las máquinas virtuales cumple con las características especificadas en la sección 2 y ejecutan el sistema operativo Linux.

Una captura de las características de cada una de las máquinas virtuales se muestra en la Figura 2.72 y en la Figura 2.73.

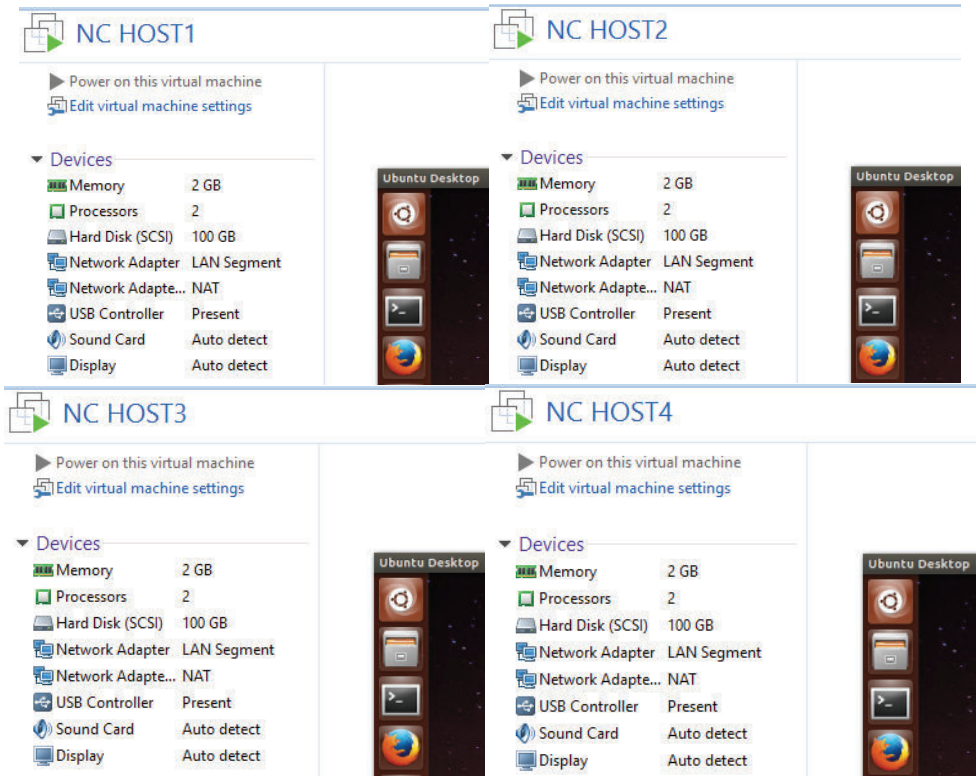


Figura 2.72 Características de las máquinas virtuales: **NC Host1**, **NC Host2**, **NC Host3** y **NC Host4**

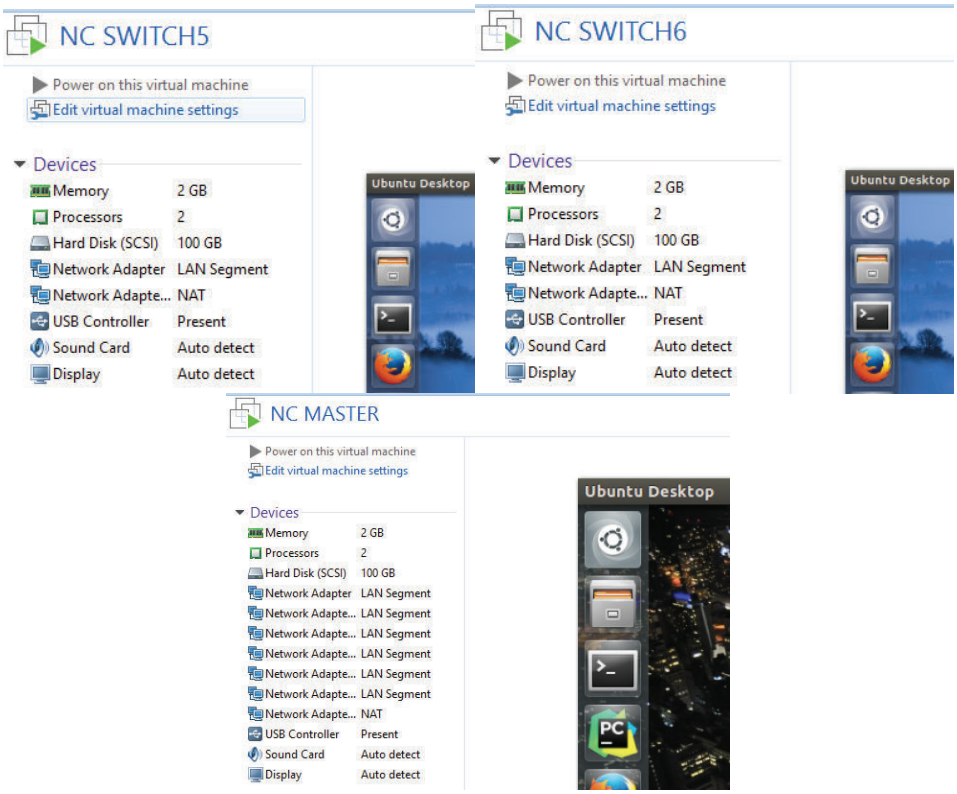


Figura 2.73 Características de las máquinas virtuales: **NC Switch5**, **NC Switch6** y **NC Master**

Una vez que cada una de las máquinas virtuales se han iniciado se comprueba que todas se puedan comunicar con **NC Master** para garantizar que los enlaces funcionan correctamente.

Un resumen de las direcciones IP empleadas en la red y los enlaces existentes se presenta en la Tabla 2.4.

Tabla 2.4 Direccionamiento y Enlaces de máquinas virtuales

Enlace	Dirección IP en NC Master	Dirección IP en Terminal
NC Master - NC Host1	192.168.10.10	192.168.10.11
NC Master - NC Host2	192.168.20.20	192.168.20.22
NC Master - NC Host3	192.168.30.30	192.168.30.33
NC Master - NC Host4	192.168.40.40	192.168.40.44
NC Master - NC Switch5	192.168.50.50	192.168.50.55
NC Master - NC Switch6	192.168.60.60	192.168.60.66

Se utiliza el comando `ping` para verificar conectividad entre cada máquina virtual con **NC Master**. Los resultados de esta prueba se muestran en la Figura 2.74.

```
ncmaster@ubuntu:~$ ping -c 1 -I 192.168.10.10 192.168.10.11
PING 192.168.10.11 (192.168.10.11) from 192.168.10.10 : 56(84) bytes of data.
64 bytes from 192.168.10.11: icmp_seq=1 ttl=64 time=0.640 ms

--- 192.168.10.11 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.640/0.640/0.640/0.000 ms
ncmaster@ubuntu:~$ ping -c 1 -I 192.168.20.20 192.168.20.22
PING 192.168.20.22 (192.168.20.22) from 192.168.20.20 : 56(84) bytes of data.
64 bytes from 192.168.20.22: icmp_seq=1 ttl=64 time=0.532 ms

--- 192.168.20.22 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.532/0.532/0.532/0.000 ms
ncmaster@ubuntu:~$ ping -c 1 -I 192.168.30.30 192.168.30.33
PING 192.168.30.33 (192.168.30.33) from 192.168.30.30 : 56(84) bytes of data.
64 bytes from 192.168.30.33: icmp_seq=1 ttl=64 time=0.517 ms

--- 192.168.30.33 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.517/0.517/0.517/0.000 ms
ncmaster@ubuntu:~$ ping -c 1 -I 192.168.40.40 192.168.40.44
PING 192.168.40.44 (192.168.40.44) from 192.168.40.40 : 56(84) bytes of data.
64 bytes from 192.168.40.44: icmp_seq=1 ttl=64 time=0.623 ms

--- 192.168.40.44 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.623/0.623/0.623/0.000 ms
ncmaster@ubuntu:~$ ping -c 1 -I 192.168.50.50 192.168.50.55
PING 192.168.50.55 (192.168.50.55) from 192.168.50.50 : 56(84) bytes of data.
64 bytes from 192.168.50.55: icmp_seq=1 ttl=64 time=0.437 ms

--- 192.168.50.55 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.437/0.437/0.437/0.000 ms
ncmaster@ubuntu:~$ ping -c 1 -I 192.168.60.60 192.168.60.66
PING 192.168.60.66 (192.168.60.66) from 192.168.60.60 : 56(84) bytes of data.
64 bytes from 192.168.60.66: icmp_seq=1 ttl=64 time=0.503 ms

--- 192.168.60.66 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.503/0.503/0.503/0.000 ms
```

Figura 2.74 Prueba de Conectividad de máquinas virtuales con **NC Master**

2.7.2. TABLAS ARP Y SINCRONIZACIÓN DE LAS MÁQUINAS VIRTUALES

Como se especificó anteriormente, las reglas del controlador SDN impiden la transmisión de tramas con dirección MAC destino tipo *broadcast* (FF:FF:FF:FF:FF:FF), y debido a esto el descubrimiento de direcciones MAC a través del protocolo ARP no es posible. Por este motivo es necesario crear tablas ARP estáticas. Para esto se utiliza el comando `sudo arp -i [interfaz] -s [dirección ip] [dirección MAC]`, asignando una correspondencia entre una dirección IP y una dirección MAC en una interfaz de red de forma manual. Este comando se ejecuta en cada uno de los equipos terminales **NC Host1**, **NC Host2**,

NC Host3 y **NC Host4**, considerando que **NC Host1** debe conocer la dirección MAC de **NC Host3**, **NC Host2** debe conocer la dirección MAC de **NC Host4**, **NC Host3** debe conocer la dirección MAC de **NC Host1** y **NC Host4** debe conocer la dirección MAC de **NC Host2**. Todas las interfaces de red en las máquinas virtuales se llaman `ens33`.

Este procedimiento se muestra en la Figura 2.75 para cada terminal seguido de la ejecución del comando `sudo arp` para desplegar el estado de la tabla ARP para cada uno, donde las entradas marcadas con la bandera `CM` son aquellas ingresadas de manera manual.

```
nchost1@ubuntu:~$ sudo arp -i ens33 -s 192.168.30.33 00:00:00:00:00:33
[sudo] password for nchost1:
nchost1@ubuntu:~$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.168.2            ether   00:50:56:f8:99:d4   C                     ens34
192.168.168.254          ether   00:50:56:e5:ae:56   C                     ens34
192.168.30.33            ether   00:00:00:00:00:33   CM                    ens33
nchost2@ubuntu:~$ sudo arp -i ens33 -s 192.168.40.44 00:00:00:00:00:44
[sudo] password for nchost2:
nchost2@ubuntu:~$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.168.2            ether   00:50:56:f8:99:d4   C                     ens34
192.168.40.44            ether   00:00:00:00:00:44   CM                    ens33
192.168.168.254          ether   00:50:56:e5:ae:56   C                     ens34
nchost3@ubuntu:~$ sudo arp -i ens33 -s 192.168.10.11 00:00:00:00:00:11
[sudo] password for nchost3:
nchost3@ubuntu:~$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.168.2            ether   00:50:56:f8:99:d4   C                     ens34
192.168.10.11            ether   00:00:00:00:00:11   CM                    ens33
192.168.168.254          ether   00:50:56:e5:ae:56   C                     ens34
nchost4@ubuntu:~$ sudo arp -i ens33 -s 192.168.20.22 00:00:00:00:00:22
[sudo] password for nchost4:
nchost4@ubuntu:~$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.168.2            ether   00:50:56:f8:99:d4   C                     ens34
192.168.168.254          ether   00:50:56:e5:ae:56   C                     ens34
192.168.20.22            ether   00:00:00:00:00:22   CM                    ens33
```

Figura 2.75 Tablas ARP

Una vez que se han creado las tablas ARP en cada uno de los terminales, se debe sincronizar lo mejor posible sus relojes de sistema, esto debido a que durante el proceso de pruebas se requiere la transmisión de tramas de manera simultánea. Se puede lograr reducir las diferencias de tiempo sincronizando cada terminal con un servidor NTP, a través del comando `ntpdate` y utilizando un servidor provisto por Ubuntu cuyo URL es `ntp.ubuntu.com`. Este procedimiento reduce la diferencia de tiempo entre los relojes de cada uno de los terminales, permitiendo emular de mejor manera el envío de tramas de manera simultánea. La ejecución de este

comando en cada uno de los terminales y su resultado se muestra en la Figura 2.76.

```
nchost1@ubuntu:~$ sudo ntpdate -u ntp.ubuntu.com
30 Jan 10:53:21 ntpdate[3349]: adjust time server 91.189.91.157 offset -0.007270 sec
nchost2@ubuntu:~$ sudo ntpdate -u ntp.ubuntu.com
30 Jan 10:53:32 ntpdate[3337]: adjust time server 91.189.91.157 offset 0.002196 sec
nchost3@ubuntu:~$ sudo ntpdate -u ntp.ubuntu.com
30 Jan 10:53:43 ntpdate[3312]: adjust time server 91.189.91.157 offset -0.020428 sec
nchost4@ubuntu:~$ sudo ntpdate -u ntp.ubuntu.com
30 Jan 10:53:54 ntpdate[3315]: adjust time server 91.189.91.157 offset -0.002240 sec
```

Figura 2.76 Sincronización de terminales

2.7.3. PRUEBA DE FUNCIONAMIENTO 1: RED SIN PÉRDIDAS SIN CODIFICACIÓN DE RED

Esta prueba permite obtener datos ideales del funcionamiento en un ambiente donde no existen pérdidas y no se utiliza codificación de red, para esto se debe editar el archivo de configuración `Perdidas.ini` y establecer en 0 los valores de porcentaje de pérdidas.

Una vez realizado este cambio, se ejecuta el comando `sudo python [dirección del archivo]` para ejecutar el archivo de Python que crea la red virtual usando Mininet. Este procedimiento y su resultado se muestra en la Figura 2.77.

```
ncmaster@ubuntu:~$ sudo python ./Dropbox/Tesis/NC\ Tesis/TopologiaIdeal.py
[sudo] password for ncmaster:
Connecting to remote controller at 127.0.0.1:6653
*** Creating network
*** Adding controller
*** Adding hosts:

*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(100.00Mbit 20ms delay) (100.00Mbit 20ms delay) (s1, s3) (100.00Mbit 1ms de
lay) (100.00Mbit 1ms delay) (s1, s5) (100.00Mbit 20ms delay) (100.00Mbit 20
ms delay) (s2, s4) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (s2, s5) (
100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (s3, s6) (100.00Mbit 1ms delay
) (100.00Mbit 1ms delay) (s4, s6) (100.00Mbit 30ms delay) (100.00Mbit 30ms
delay) (s5, s6)
*** Configuring hosts

*** Adding hardware interface ens33 to switch s1
*** Adding hardware interface ens34 to switch s2
*** Adding hardware interface ens35 to switch s3
*** Adding hardware interface ens36 to switch s4
*** Adding hardware interface ens36 to switch s5
*** Adding hardware interface ens36 to switch s6
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...(100.00Mbit 20ms delay) (100.00Mbit 1ms delay) (100.00
Mbit 20ms delay) (100.00Mbit 1ms delay) (100.00Mbit 20ms delay) (100.00Mbit
 1ms delay) (100.00Mbit 20ms delay) (100.00Mbit 1ms delay) (100.00Mbit 1ms
delay) (100.00Mbit 1ms delay) (100.00Mbit 30ms delay) (100.00Mbit 1ms delay
) (100.00Mbit 1ms delay) (100.00Mbit 30ms delay)
*** Starting CLI:
mininet>
```

Figura 2.77 Ejecución de red con topología *butterfly* sin pérdidas

Una vez levantada la red virtual se ejecuta el comando `PYTHONPATH=. ./bin/ryu-manager [ubicación del archivo]`, el cual inicia la aplicación y el controlador Ryu. Para esta prueba se utiliza la aplicación sin codificación de red la cual instala reglas en los *switches* **s1**, **s2**, **s3** y **s4** como se muestra en la Figura 2.78.

```
ncmaster@ubuntu:~$ cd ryu/
ncmaster@ubuntu:~/ryu$ PYTHONPATH=. ./bin/ryu-manager ~/Dropbox/Tesis/NC\ Tesis/Regu
larController.py
loading app /home/ncmaster/Dropbox/Tesis/NC Tesis/RegularController.py
loading app ryu.controller.ofp_handler
instantiating app /home/ncmaster/Dropbox/Tesis/NC Tesis/RegularController.py of Swit
chNC
instantiating app ryu.controller.ofp_handler of OFPHandler
-------------RED CONFIGURADA CORRECTAMENTE-------------
-------------------------------------------------------
|           ||         MATCH      ||      ACTIONS     ||
| DATAPATH  ||-----------------------------------------||
|           || INPORT | SRC | DST || MOD_DST | OUTPUT ||
-------------------------------------------------------
|     1     ||    1   | :11 | :33 ||   NONE  |    2   ||
|     1     ||    2   | :33 | :11 ||   NONE  |    1   ||
-------------------------------------------------------
|     2     ||    1   | :22 | :44 ||   NONE  |    2   ||
|     2     ||    2   | :44 | :22 ||   NONE  |    1   ||
-------------------------------------------------------
|     3     ||    1   | :33 | :11 ||   NONE  |    2   ||
|     3     ||    2   | :11 | :33 ||   NONE  |    1   ||
-------------------------------------------------------
|     4     ||    1   | :44 | :22 ||   NONE  |    2   ||
|     4     ||    2   | :22 | :44 ||   NONE  |    1   ||
-------------------------------------------------------
```

Figura 2.78 Controlador sin Codificación de Red

Con la red virtual, el controlador y la aplicación para generar reglas ejecutándose, se procede a realizar pruebas enviando mensajes ICMP usando el comando `ping` desde los terminales **NC Host3** y **NC Host4**. Para esto se crearon dos *scripts* (`ICMP_Host3.sh` y `ICMP_Host4.sh`), en los cuales se especifica el tamaño del paquete a enviar con el parámetro `-s`, la interfaz de red desde donde se enviarán las tramas con el parámetro `-I`, la cantidad de paquetes que se enviarán con el parámetro `-c`, y la dirección IP de destino. Los resultados de la ejecución de este comando se almacenan en un archivo de texto en cada terminal llamado `resultados.txt`. El contenido de estos *scripts* se muestra en la Figura 2.79 y en la Figura 2.80, en las cuales se trasmiten 10 paquetes de 100 MB cada uno.

```
#! /bin/bash

echo $(ping -s 100 -I 192.168.30.33 -c 10 192.168.10.11) > resultados.txt
```

Figura 2.79 *Script* `ICMP_Host3.sh`

```
#! /bin/bash

echo $(ping -s 100 -I 192.168.40.44 -c 10 192.168.20.22) > resultados.txt
```

Figura 2.80 *Script* `ICMP_Host4.sh`

Estos *scripts* se ejecutan usando el comando `at [hora] -f [script a ejecutar]`. El comando `at` ejecuta un *script* definido a una hora específica. Esto permite sincronizar de mejor manera el envío de tramas. La ejecución de este comando en cada terminal se muestra en la Figura 2.81.

```
nchost3@ubuntu:~$ at 11:40 -f ~/Dropbox/Tesis/Pruebas/ICMP_Host3.sh
warning: commands will be executed using /bin/sh
job 86 at Mon Jan 30 11:40:00 2017
nchost4@ubuntu:~$ at 11:40 -f ~/Dropbox/Tesis/Pruebas/ICMP_Host4.sh
warning: commands will be executed using /bin/sh
job 83 at Mon Jan 30 11:40:00 2017
```

Figura 2.81 Ejecución de *scripts* con comando `at`

Utilizando Wireshark se puede comprobar la transmisión de las tramas, y una vez que se termina la ejecución del *script* los archivos de `resultados.txt` permiten conocer los resultados de la ejecución del comando en cada máquina virtual.

En la Figura 2.82 se muestra el resultado de la captura de Wireshark y el contenido del archivo `resultados.txt` para **NC Host3**. Para verificar la cantidad de tramas trasmitidas se verifica los números de secuencia, ya que la comprobación que realiza Wireshark en la última columna de datos puede ser errónea.

Source	Destination	Protocol	Length	Info
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=1/256, ttl=64 (reply in 2)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=1/256, ttl=64 (request in 1)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=2/512, ttl=64 (no response found!)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=2/512, ttl=64 (request in 3)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=3/768, ttl=64 (reply in 6)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=3/768, ttl=64 (request in 5)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=4/1024, ttl=64 (reply in 8)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=4/1024, ttl=64 (request in 7)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=5/1280, ttl=64 (reply in 10)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=5/1280, ttl=64 (request in 9)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=6/1536, ttl=64 (reply in 12)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=6/1536, ttl=64 (request in 11)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=7/1792, ttl=64 (reply in 14)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=7/1792, ttl=64 (request in 13)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=8/2048, ttl=64 (reply in 16)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=8/2048, ttl=64 (request in 15)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=9/2304, ttl=64 (reply in 18)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=9/2304, ttl=64 (request in 17)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0dcc, seq=10/2560, ttl=64 (reply in 20)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0dcc, seq=10/2560, ttl=64 (request in 19)

resultados.txt (~/) - gedit

Open Save

```
PING 192.168.10.11 (192.168.10.11) from 192.168.30.33 : 100
(128) bytes of data. 108 bytes from 192.168.10.11: icmp_seq=1
ttl=64 time=41.4 ms 108 bytes from 192.168.10.11: icmp_seq=2
ttl=64 time=41.6 ms 108 bytes from 192.168.10.11: icmp_seq=3
ttl=64 time=42.4 ms 108 bytes from 192.168.10.11: icmp_seq=4
ttl=64 time=40.8 ms 108 bytes from 192.168.10.11: icmp_seq=5
ttl=64 time=41.2 ms 108 bytes from 192.168.10.11: icmp_seq=6
ttl=64 time=41.4 ms 108 bytes from 192.168.10.11: icmp_seq=7
ttl=64 time=40.8 ms 108 bytes from 192.168.10.11: icmp_seq=8
ttl=64 time=41.3 ms 108 bytes from 192.168.10.11: icmp_seq=9
ttl=64 time=41.3 ms 108 bytes from 192.168.10.11: icmp_seq=10
ttl=64 time=41.4 ms --- 192.168.10.11 ping statistics --- 10
packets transmitted, 10 received, 0% packet loss, time 9050ms
rtt min/avg/max/mdev = 40.810/41.385/42.412/0.502 ms
```

Figura 2.82 Resultados de prueba en **NC Host3**

Estos resultados comprueban el correcto funcionamiento tanto de la red así como del controlador en un ambiente sin pérdidas.

2.7.4. PRUEBA DE FUNCIONAMIENTO 2: RED CON PÉRDIDAS Y SIN CODIFICACIÓN DE RED

Esta prueba permite obtener datos de funcionamiento en un ambiente donde existe un 50% de probabilidad de perder tramas durante la transmisión y no se utiliza codificación de red.

Para esto se debe editar el archivo de configuración `Perdidas.ini` y establecer en 50 los valores de porcentaje de pérdidas.

La ejecución de esta red virtual y su resultado se muestran en la Figura 2.83 donde se resalta el porcentaje de pérdidas configurado.

```
ncmaster@ubuntu:~$ sudo python ./Dropbox/Tesis/NC\ Tesis/TopologiaReal.py celar
Connecting to remote controller at 127.0.0.1:6653
*** Creating network
*** Adding controller
*** Adding hosts:

*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(100.00Mbit 10ms delay 50.00000% loss) (100.00Mbit 10ms delay 50.00000% loss) (s1, s
3) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (s1, s5) (100.00Mbit 10ms delay 50.
00000% loss) (100.00Mbit 10ms delay 50.00000% loss) (s2, s4) (100.00Mbit 1ms delay)
(100.00Mbit 1ms delay) (s2, s5) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (s3, s
6) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (s4, s6) (100.00Mbit 20ms delay) (1
00.00Mbit 20ms delay) (s5, s6)
*** Configuring hosts

*** Adding hardware interface ens33 to switch s1
*** Adding hardware interface ens34 to switch s2
*** Adding hardware interface ens35 to switch s3
*** Adding hardware interface ens36 to switch s4
*** Adding hardware interface ens36 to switch s5
*** Adding hardware interface ens36 to switch s6
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...(100.00Mbit 10ms delay 50.00000% loss) (100.00Mbit 1ms delay) (
100.00Mbit 10ms delay 50.00000% loss) (100.00Mbit 1ms delay) (100.00Mbit 10ms delay
50.00000% loss) (100.00Mbit 1ms delay) (100.00Mbit 10ms delay 50.00000% loss) (100.0
0Mbit 1ms delay) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (100.00Mbit 20ms dela
y) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (100.00Mbit 20ms delay)
*** Starting CLI:
mininet>
```

Figura 2.83 Ejecución de la red con topología *butterfly* con pérdidas

El controlador y los *scripts* en **NC Host3** y **NC Host4** se ejecutan de la misma manera que en la prueba anterior. Una vez que la prueba ha terminado, el resultado obtenido tanto en Wireshark como en el archivo `resultados.txt` para **NC Host3** se muestra en la Figura 2.84.

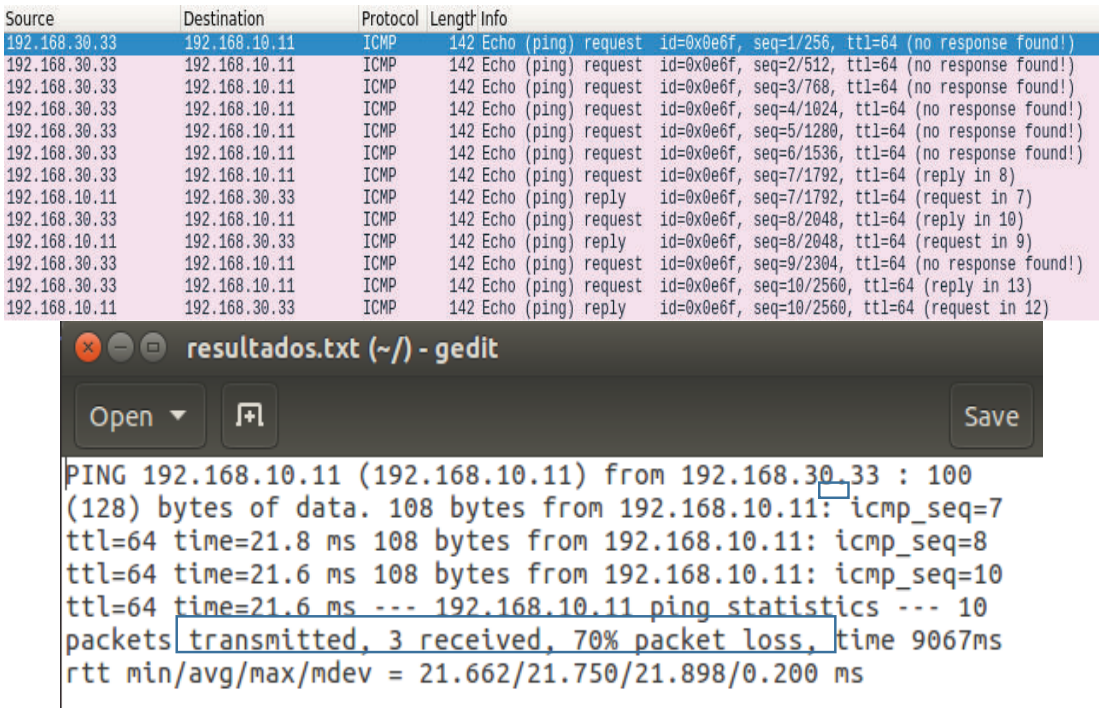


Figura 2.84 Resultados de prueba en **NC Host3**

En la prueba presentada el valor de porcentaje de pérdidas es del 70% para **NC Host3**, esto debido a que la prueba se realiza únicamente con 10 paquetes. Con la ayuda de Wireshark se aprecia que únicamente los paquetes 7, 8 y 10 se transmitieron sin pérdidas.

Un análisis más detallado del porcentaje de pérdidas se muestra en el Capítulo 3.

2.7.5. PRUEBA DE FUNCIONAMIENTO 3: RED SIN PÉRDIDAS Y CON CODIFICACIÓN DE RED

Esta prueba permite obtener datos de funcionamiento en un ambiente donde no existen tramas perdidas durante la transmisión, pero si se utiliza codificación de red, para esto se debe editar el archivo de configuración `Perdidas.ini` y establecer en 0 los valores de porcentaje de pérdidas.

Se utiliza la aplicación que instala las reglas necesarias para la codificación de red. Al ejecutar esta aplicación el controlador instalará reglas en los 6 *switches* presentes en la red virtual, y se obtendrá como resultado lo mostrado en la Figura 2.85.

```
ncmaster@ubuntu:~/ryu$ PYTHONPATH=. ./bin/ryu-manager ~/Dropbox/Tesis/NC\ Tesis/Netw
orkCodingController.py
loading app /home/ncmaster/Dropbox/Tesis/NC Tesis/NetworkCodingController.py
loading app ryu.controller.ofp_handler
instantiating app /home/ncmaster/Dropbox/Tesis/NC Tesis/NetworkCodingController.py o
f SwitchNC
instantiating app ryu.controller.ofp_handler of OFPHandler
-------------RED CONFIGURADA CORRECTAMENTE------------
```

	MATCH			ACTIONS	
DATAPATH	INPORT	SRC	DST	MOD_DST	OUTPUT
1	1	:11	:33	NONE	3
1	2	:33	:11	NONE	3
1	3	:11	:33	NONE	2
1	3	:33	:11	NONE	1
2	1	:22	:44	NONE	3
2	2	:44	:22	NONE	3
2	3	:22	:44	NONE	2
2	3	:44	:22	NONE	1
3	1	:33	:11	NONE	3
3	2	:11	:33	NONE	3
3	3	:33	:11	NONE	2
3	3	:11	:33	NONE	1
4	1	:44	:22	NONE	3
4	2	:22	:44	NONE	3
4	3	:44	:22	NONE	2
4	3	:22	:44	NONE	1
5	1	:11	:33	:55	4
5	2	:22	:44	:55	4
5	1	:33	:11	:55	4
5	2	:44	:22	:55	4
5	3	:77	:88	:55	4
5	4	:11	:33	NONE	1
5	4	:22	:44	NONE	2
5	4	:33	:11	NONE	1

Figura 2.85 Controlador con Codificación de Red

Una vez iniciada la red virtual y el controlador se ejecutan las aplicaciones para codificación y decodificación instaladas en los terminales **NC Switch5** y **NC Switch6**. Como resultado de esto se desplegará en cada terminal la información presentada en la Figura 2.86.

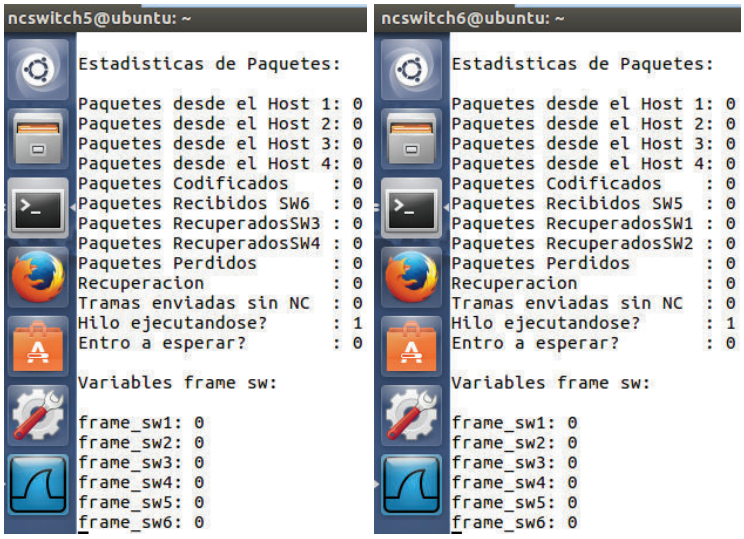


Figura 2.86 Ejecución de las aplicaciones de codificación de red en **NC Switch5** y **NC Switch6**

Ahora se puede ejecutar el comando para realizar las pruebas en **NC Host3** y **NC Host4** de la misma manera que en las anteriores pruebas.

Al terminar la ejecución de la prueba se obtendrán los valores desplegados para **NC Switch5** y **NC Switch6** como se presenta en la Figura 2.87.

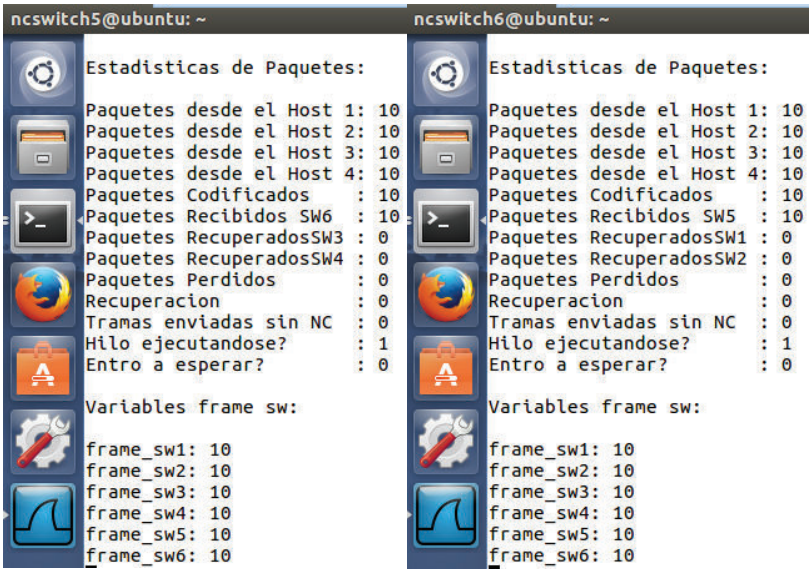
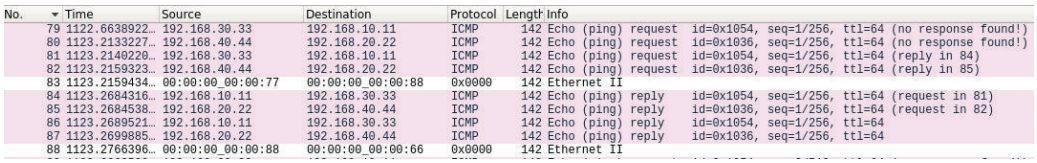


Figura 2.87 Resultados de prueba en **NC Switch5** y **NC Switch6**

Como se puede apreciar en la Figura 2.87, se generaron 10 tramas en total desde cada terminal, se generaron además 10 tramas codificadas en **NC Switch5** y 10 tramas codificadas en **NC Switch6,** las cuales debido a las condiciones ideales de

la prueba no fueron empleadas para recuperar tramas. Tampoco existe tramas perdidas y ninguna trama viajó sin ser utilizada en el proceso de codificación.

Estos valores se pueden comprobar con los datos capturados por Wireshark. En la Figura 2.88 se presenta los datos de **NC Switch6** para el primer paquete ICMP enviado en la prueba.

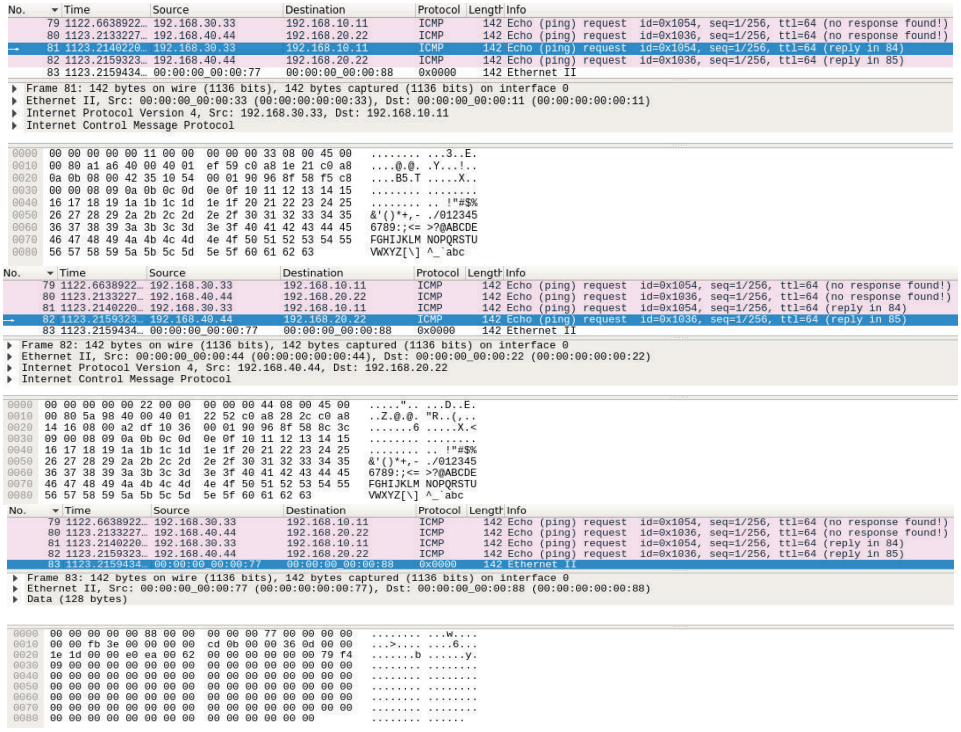


No.	Time	Source	Destination	Protocol	Length	Info
79	1122.6638922…	192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x1054, seq=1/256, ttl=64 (no response found!)
80	1123.2133227…	192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x1036, seq=1/256, ttl=64 (no response found!)
81	1123.2140220…	192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x1054, seq=1/256, ttl=64 (reply in 84)
82	1123.2159323…	192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x1036, seq=1/256, ttl=64 (reply in 85)
83	1123.2159434…	00:00:00_00:00:77	00:00:00_00:00:88	0x0000	142	Ethernet II
84	1123.2684316…	192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x1054, seq=1/256, ttl=64 (request in 81)
85	1123.2684538…	192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x1036, seq=1/256, ttl=64 (request in 82)
86	1123.2689521…	192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x1054, seq=1/256, ttl=64
87	1123.2699885…	192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x1036, seq=1/256, ttl=64
88	1123.2766396…	00:00:00_00:00:88	00:00:00_00:00:66	0x0000	142	Ethernet II

Figura 2.88 Resultados de Wireshark para primer paquete ICMP de prueba en **NC Switch6**

La trama número 79 fue generada en **NC Host3**, el número 80 fue generado en **NC Host4,** una vez que ambas tramas ingresan y son procesadas por la aplicación son reenviadas a la red (paquetes 81 y 82).

Después se envía la trama codificada (trama 83). En la Figura 2.89 se muestran las tramas 81, 82 y 83, y se puede comprobar que su contenido es el resultado de una operación XOR entre estas dos tramas.



No.	Time	Source	Destination	Protocol	Length	Info
79	1122.6638922…	192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x1054, seq=1/256, ttl=64 (no response found!)
80	1123.2133227…	192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x1036, seq=1/256, ttl=64 (no response found!)
81	1123.2140220…	192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x1054, seq=1/256, ttl=64 (reply in 84)
82	1123.2159323…	192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x1036, seq=1/256, ttl=64 (reply in 85)
83	1123.2159434…	00:00:00_00:00:77	00:00:00_00:00:88	0x0000	142	Ethernet II

```
▸ Frame 81: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:33 (00:00:00:00:00:33), Dst: 00:00:00_00:00:11 (00:00:00:00:00:11)
▸ Internet Protocol Version 4, Src: 192.168.30.33, Dst: 192.168.10.11
▸ Internet Control Message Protocol
```

```
0000  00 00 00 00 00 11 00 00  00 00 00 33 08 00 45 00   ....... ...3..E.
0010  00 80 a1 a6 40 00 40 01  ef 59 c0 a8 1e 21 c0 a8   ....@.@. .Y...!..
0020  0a 0b 08 00 42 35 10 54  00 01 90 96 8f 58 f5 c8   ....B5.T .....X..
0030  00 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ........ ........
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ........ .. !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37 38 39 3a 3b 3c 3d  3e 3f 40 41 42 43 44 45   6789:;<= >?@ABCDE
0070  46 47 48 49 4a 4b 4c 4d  4e 4f 50 51 52 53 54 55   FGHIJKLM NOPQRSTU
0080  56 57 58 59 5a 5b 5c 5d  5e 5f 60 61 62 63          VWXYZ[\] ^_`abc
```

No.	Time	Source	Destination	Protocol	Length	Info
79	1122.6638922…	192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x1054, seq=1/256, ttl=64 (no response found!)
80	1123.2133227…	192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x1036, seq=1/256, ttl=64 (no response found!)
81	1123.2140220…	192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x1054, seq=1/256, ttl=64 (reply in 84)
82	1123.2159323…	192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x1036, seq=1/256, ttl=64 (reply in 85)
83	1123.2159434…	00:00:00_00:00:77	00:00:00_00:00:88	0x0000	142	Ethernet II

```
▸ Frame 82: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:44 (00:00:00:00:00:44), Dst: 00:00:00_00:00:22 (00:00:00:00:00:22)
▸ Internet Protocol Version 4, Src: 192.168.40.44, Dst: 192.168.20.22
▸ Internet Control Message Protocol
```

```
0000  00 00 00 00 00 22 00 00  00 00 00 44 08 00 45 00   ....."·· ···D··E·
0010  00 80 5a 98 40 00 40 01  22 52 c0 a8 28 2c c0 a8   ··Z·@·@· "R··(,··
0020  14 16 08 00 a2 df 10 36  00 01 90 96 8f 58 8c 3c   ·······6 ·····X·<
0030  09 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ········ ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345
0060  36 37 38 39 3a 3b 3c 3d  3e 3f 40 41 42 43 44 45   6789:;<= >?@ABCDE
0070  46 47 48 49 4a 4b 4c 4d  4e 4f 50 51 52 53 54 55   FGHIJKLM NOPQRSTU
0080  56 57 58 59 5a 5b 5c 5d  5e 5f 60 61 62 63          VWXYZ[\] ^_`abc
```

No.	Time	Source	Destination	Protocol	Length	Info
79	1122.6638922…	192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x1054, seq=1/256, ttl=64 (no response found!)
80	1123.2133227…	192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x1036, seq=1/256, ttl=64 (no response found!)
81	1123.2140220…	192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x1054, seq=1/256, ttl=64 (reply in 84)
82	1123.2159323…	192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x1036, seq=1/256, ttl=64 (reply in 85)
83	1123.2159434…	00:00:00_00:00:77	00:00:00_00:00:88	0x0000	142	Ethernet II

```
▸ Frame 83: 142 bytes on wire (1136 bits), 142 bytes captured (1136 bits) on interface 0
▸ Ethernet II, Src: 00:00:00_00:00:77 (00:00:00:00:00:77), Dst: 00:00:00_00:00:88 (00:00:00:00:00:88)
▸ Data (128 bytes)
```

```
0000  00 00 00 00 00 88 00 00  00 00 00 77 00 00 00 00   ........ ...w....
0010  00 00 fb 3e 00 00 00 00  cd 0b 00 00 36 0d 00 00   ...>.... ....6...
0020  1e 1d 00 00 e0 ea 00 62  00 00 00 00 00 00 79 f4   .......b ......y.
0030  09 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
```

Figura 2.89 Verificación de tramas de **NC Switch6** en Wireshark

De la misma manera, las tramas desde el 84 al 87 son aquellos recibidos desde **NC Host1** y **NC Host2**, los cuales son la respuesta a los paquetes de **NC Host3** y **NC Host4**.

La trama número 88 es la trama codificada y generada en **NC Switch5.** En la Figura 2.90 se puede comprobar que la trama 88 es el resultado de una operación XOR entre las tramas 86 y 87.

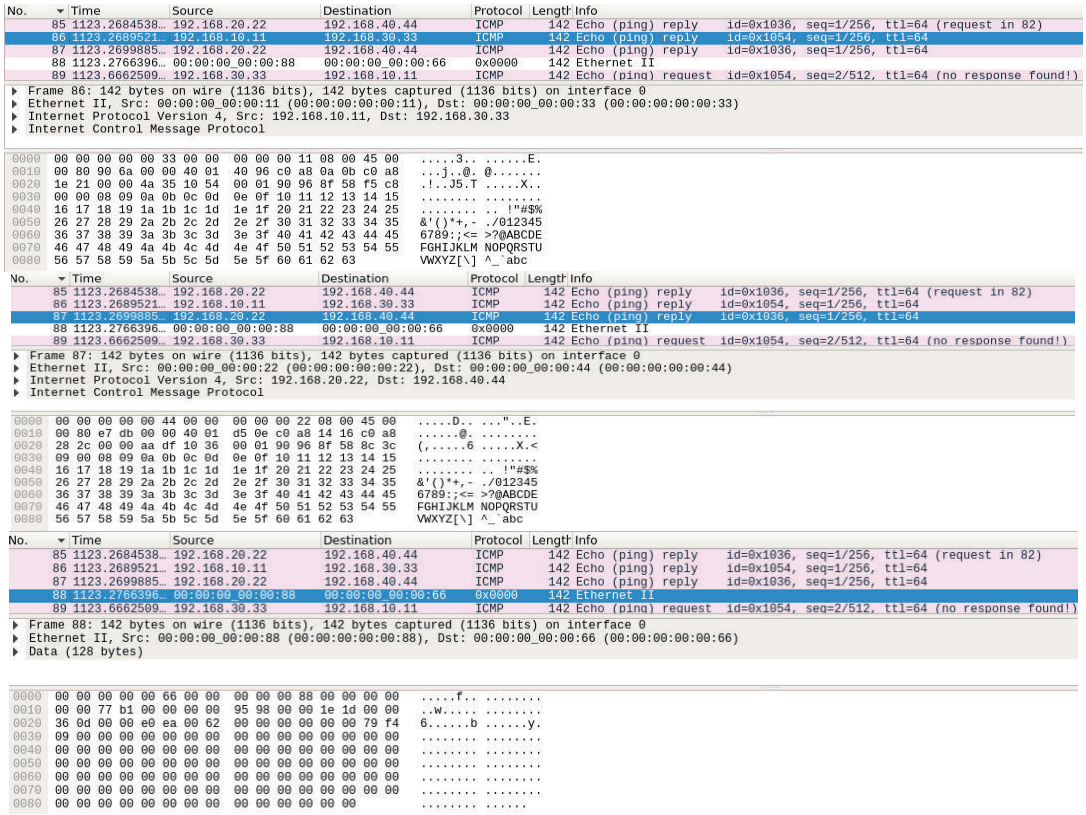


Figura 2.90 Verificación de tramas de **NC Switch5** en Wireshark

Con esto se comprueba el proceso de codificación en **NC Switch5** y **NC Switch6**.

2.7.6. PRUEBA DE FUNCIONAMIENTO 4: RED CON PÉRDIDAS CON CODIFICACIÓN DE RED

Para esta prueba se ejecutará la red virtual con una probabilidad de pérdida del 10% en los enlaces entre **s1** y **s3** y entre **s2** y **s4**, para esto se debe editar el archivo de configuración `Perdidas.ini` y establecer en 10 los valores de porcentaje de pérdidas.

Al terminar la prueba **NC Switch5** y **NC Switch6** presentas las estadísticas presentadas en la Figura 2.91.

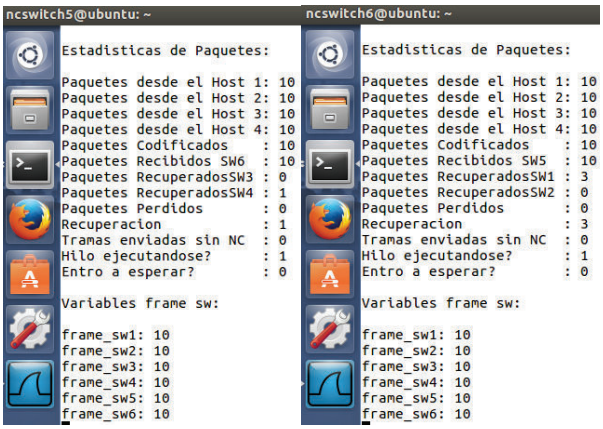


Figura 2.91 Resultados de prueba en **NC Switch5** y **NC Switch6**

Como se puede apreciar, se recuperaron en total 4 tramas, 1 en **NC Switch5** y 3 en **NC Switch6**.

El proceso de recuperación consiguió que tanto **NC Host3** y **NC Host4** no presenten ninguna pérdida, como se puede comprobar en sus archivos de `resultados.txt`, como se muestra en la Figura 2.92 y en la Figura 2.93.

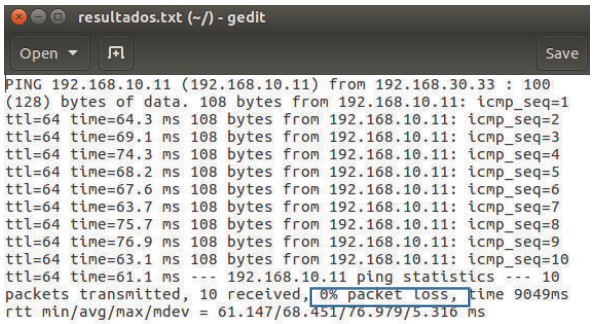


Figura 2.92 Resultados de prueba en **NC Host3**

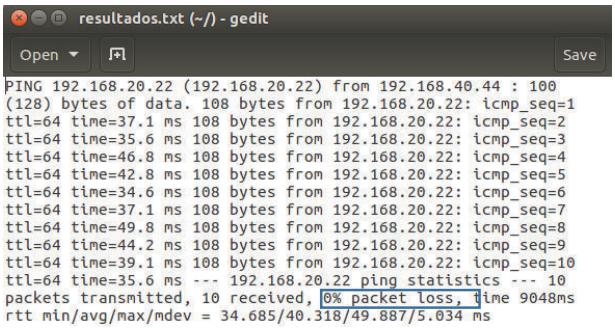


Figura 2.93 Resultados de prueba en **NC Host4**

De la misma manera, las capturas de Wireshark en **NC Host1** y **NC Host2** presentadas en las Figuras 2.94 y 2.95, respectivamente, no indican la ausencia de tramas. Lo cual comprueba que el proceso de codificación de red es transparente para los equipos terminales.

Source	Destination	Protocol	Length	Info
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=1/256, ttl=64 (reply in 15)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=1/256, ttl=64 (request in 14)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=2/512, ttl=64 (reply in 17)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=2/512, ttl=64 (request in 16)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=3/768, ttl=64 (reply in 19)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=3/768, ttl=64 (request in 18)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=4/1024, ttl=64 (reply in 21)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=4/1024, ttl=64 (request in 20)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=5/1280, ttl=64 (reply in 23)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=5/1280, ttl=64 (request in 22)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=6/1536, ttl=64 (reply in 25)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=6/1536, ttl=64 (request in 24)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=7/1792, ttl=64 (reply in 27)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=7/1792, ttl=64 (request in 26)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=8/2048, ttl=64 (reply in 29)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=8/2048, ttl=64 (request in 28)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=9/2304, ttl=64 (reply in 31)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=9/2304, ttl=64 (request in 30)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x11f8, seq=10/2560, ttl=64 (reply in 33)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x11f8, seq=10/2560, ttl=64 (request in 32)

Figura 2.94 Captura de Wireshark en **NC Host1**

Source	Destination	Protocol	Length	Info
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=1/256, ttl=64 (reply in 14)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=1/256, ttl=64 (request in 13)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=2/512, ttl=64 (reply in 16)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=2/512, ttl=64 (request in 15)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=3/768, ttl=64 (reply in 18)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=3/768, ttl=64 (request in 17)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=4/1024, ttl=64 (reply in 21)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=4/1024, ttl=64 (request in 20)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=5/1280, ttl=64 (reply in 23)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=5/1280, ttl=64 (request in 22)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=6/1536, ttl=64 (reply in 25)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=6/1536, ttl=64 (request in 24)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=7/1792, ttl=64 (reply in 27)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=7/1792, ttl=64 (request in 26)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=8/2048, ttl=64 (reply in 29)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=8/2048, ttl=64 (request in 28)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=9/2304, ttl=64 (reply in 31)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=9/2304, ttl=64 (request in 30)
192.168.40.44	192.168.20.22	ICMP	142	Echo (ping) request id=0x12ef, seq=10/2560, ttl=64 (reply in 33)
192.168.20.22	192.168.40.44	ICMP	142	Echo (ping) reply id=0x12ef, seq=10/2560, ttl=64 (request in 32)

Figura 2.95 Captura de Wireshark en **NC Host2**

2.7.7. PRUEBA DE FUNCIONAMIENTO 5: ENVÍO DE TRAMAS SIN USAR CODIFICACIÓN DE RED

La última prueba se realiza para verificar el reenvío de una trama en caso de no existir otra trama con la cual generar una trama codificada. Para esto se ejecutará el mismo escenario anterior, pero solo se ejecutará la prueba entre **NC Host3** y **NC Host1**. Al finalizar la prueba se muestran las estadísticas obtenidas en **NC Switch5** y **NC Swhtch6** en la Figura 2.96.

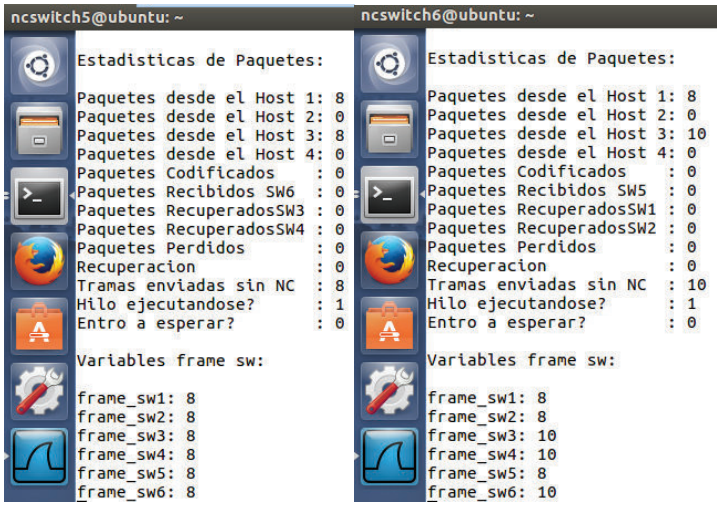


Figura 2.96 Resultados de prueba en **NC Switch5** y **NC Switch6**

Como se puede apreciar el programa funciona de manera correcta en caso que solo dos de los terminales participen en la transmisión, sin embargo, el proceso de codificación de red no puede realizarse y por lo tanto existe pérdida de dos tramas enviadas desde **NC Host3** y que no llegaron a **NC Host1**, esto genera un porcentaje de pérdidas del 20%, como se muestra en el archivo de resultados de la Figura 2.97.

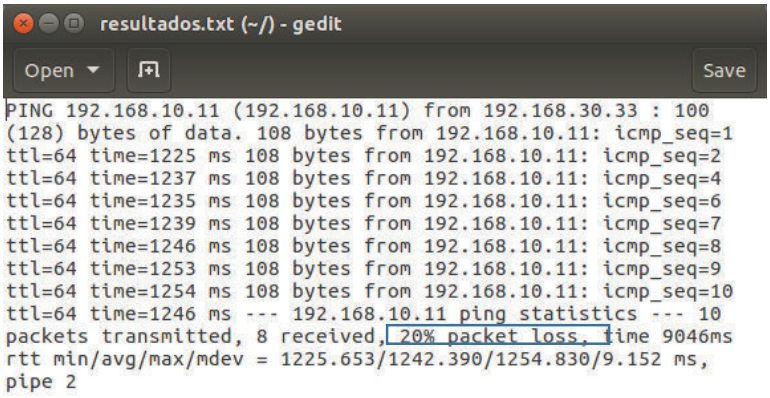


Figura 2.97 Resultados de prueba en **NC Host3**

Una vez verificado el funcionamiento de las aplicaciones desarrolladas se realizaron múltiples pruebas adicionales con el fin de demostrar que el proceso de codificación de red contribuye a reducir porcentaje de pérdidas en la red. Los resultados de estas pruebas y su discusión se presentan en el Capítulo 3.

CAPÍTULO 3. RESULTADOS Y DISCUSIÓN

En este capítulo se presentan los resultados de las distintas pruebas realizadas en el ambiente de red implementado. Se detallan las características de las pruebas: número de tramas enviadas, número de pruebas realizadas, configuración del porcentaje de pérdidas y los porcentajes de pérdidas obtenidos.

3.1. RESULTADOS DE LAS PRUEBAS

Se realizaron un total de 124 pruebas en donde se envían un total de 100 tramas de datos desde cada terminal con las siguientes consideraciones:

- 47 pruebas sin codificación de red y 47 pruebas con codificación de red.
- Para cada grupo de 47 pruebas se toman dos para pruebas con pérdidas del 0 y 100% y el resto se dividen en 9 grupos de 5, para tener 5 pruebas con distintos porcentajes de pérdidas, empezando en 10% y terminando en 90%.
- Se realizaron 10 pruebas en donde el enlace entre **s1** y **s3** no posee pérdidas y el enlace entre **s2** y **s4** varía entre 10 y 100% en pasos de 10%.
- Se realizaron 10 pruebas en donde el enlace entre **s2** y **s4** no posee pérdidas y el enlace entre **s1** y **s3** varía entre 10 y 100% en pasos de 10%.

El cálculo del porcentaje de pérdidas se realiza en base al número de tramas totales que circulan por los enlaces en cada prueba. Considerando que cada terminal produce 100 tramas por prueba, el número total de tramas en cada enlace es 200 y en base a este total, utilizando el programa Wireshark para conocer el número de tramas que circularon durante la prueba, se conoce el número total de tramas perdidas durante la transmisión.

3.1.1. RESULTADOS DE PRUEBAS SIN CODIFICACIÓN DE RED

La Tabla 3.1 presenta los resultados de las pruebas realizadas en un ambiente sin codificación de red.

En cada prueba se envía un total de 100 paquetes ICMP los cuales posen un peso de 100 MB cada uno.

Se presentan los porcentajes de pérdidas configurados dentro del archivo de configuración de la topología, los porcentajes de pérdidas obtenidos en los enlaces entre **s1** y **s3** y entre **s2** y **s4**, y los valores promedio de estos porcentajes cada 5 pruebas.

Tabla 3.1 Resultados de Pruebas sin Codificación de Red

% de Pérdidas Configurado	% de Pérdidas Obtenido en el enlace s1 – s3	% de Pérdidas Obtenido Promedio en el enlace s1 – s3	% de Pérdidas Obtenido en el enlace s2 – s4	% de Pérdidas Obtenido Promedio en el enlace s2– s4
0%	0%	0%	0%	0%
10%	6%	7.6%	8.5%	5.9%
	6%		6%	
	6%		4%	
	12%		5%	
	8%		6%	
20%	16%	16.4%	20%	16%
	16%		14.5%	
	16%		18%	
	14%		14%	
	20%		13.5%	
30%	27.5%	25.5%	25.5%	27.1%
	20.5%		30.5%	
	16.5%		26.5%	
	23.5%		23.5%	
	29.5%		29.5%	
40%	34.5%	37.3%	40.5%	36.7%
	40.5%		32.5%	
	37.5%		34.5%	
	33.5%		39.5%	
	40.5%		36.5%	
50%	47.5%	46.7%	45.5%	46.5%
	43.5%		49.5%	
	52.5%		46.5%	
	51.5%		47.5%	
	38.5%		43.5%	
60%	62.5%	61.3%	51.5%	53.9%
	59.5%		55.5%	
	62.5%		58.5%	
	64.5%		55.5%	
	57.5%		48.5%	
70%	63.5%	67.5%	69.5%	68.5%
	65.5%		68.5%	
	70.5%		73.5%	
	68.5%		62.5%	
	69.5%		68.5%	
80%	67.5%	74.5%	95.5%	81.5%
	78.5%		77.5%	
	77.5%		80.5%	
	72.5%		78.5%	
	76.5%		75.5%	
90%	87.5%	87.8%	81.5%	88.3%
	87.5%		92.5%	
	84.5%		89.5%	
	91.5%		86.5%	
	88%		91.5%	
100%	100%	100%	100%	100%

3.1.2. RESULTADOS DE PRUEBAS CON CODIFICACIÓN DE RED

La Tabla 3.3 presenta los resultados de pruebas realizadas en un ambiente con codificación de red. La cantidad de paquetes enviados es igual al ambiente anterior.

3.1.3. RESULTADOS DE PRUEBAS CON UN ENLACE SIN PÉRDIDAS

Con el fin de analizar si las aplicaciones implementadas permiten la completa recuperación de información en caso de fallas en un único enlace se implementaron dos escenarios adicionales de pruebas con un solo enlace con porcentaje de pérdidas configurado. Se presentan los porcentajes de pérdidas obtenidos en base a la contabilización de tramas transmitidas por cada enlace durante cada prueba.

La Tabla 3.2 presenta los resultados de pruebas realizadas en un ambiente sin y con codificación de red donde el enlace entre **s1** y **s3** no posee pérdidas.

En cada prueba se envía un total de 200 paquetes ICMP de 100 MB cada uno. Se presenta el porcentaje de pérdidas configurado en Mininet y el obtenido en el enlace entre **s2** y **s4**.

Tabla 3.2 Resultados de Pruebas con enlace **s1** – **s3** sin Pérdidas

% de Pérdidas Configurado en el enlace s2 – s4	% de Pérdidas Obtenido en el enlace s2 – s4 sin Codificación de Red	% de Pérdidas Obtenido en el enlace s2 – s4 con Codificación de Red
10%	8.5%	0%
20%	14%	0%
30%	23.5%	0%
40%	32%	0%
50%	48%	0%
60%	51%	0%
70%	62.5%	0%
80%	77.5%	0%
90%	89.5%	0%
100%	100%	0%

La Tabla 3.4 presenta los resultados de pruebas realizadas en un ambiente sin y con codificación de red donde el enlace entre **s2** y **s4** no posee pérdidas. En cada prueba se envía un total de 200 paquetes ICMP con un peso de 100 MB cada uno. Se presenta el porcentaje de pérdidas configurado en Mininet dentro del archivo de configuración correspondiente y el obtenido durante la prueba en el enlace entre **s1** y **s3**.

Tabla 3.3 Resultados de Pruebas con Codificación de Red

% de Pérdidas Configurado	% de Pérdidas Obtenido en el enlace s1 – s3	% de Pérdidas Obtenido Promedio en el enlace s1 – s3	% de Pérdidas Obtenido en el enlace s2 – s4	% de Pérdidas Obtenido Promedio en el enlace s2– s4
0%	0%	0%	0%	0%
10%	0%	0.1%	0%	0.1%
	0%		0%	
	0%		0%	
	0.5%		0.5%	
	0%		0%	
20%	1.5%	1.3%	1.5%	1.3%
	1.5%		1.5%	
	2.5%		2.5%	
	0.5%		0.5%	
	0.5%		0.5%	
30%	5.5%	5.3%	5.5%	5.3%
	5.5%		5.5%	
	3.5%		3.5%	
	6%		6%	
	6%		6%	
40%	14%	13.7%	14%	13.7%
	13.5%		13.5%	
	13.5%		13.5%	
	12.5%		12.5%	
	15%		15%	
50%	20%	20.4%	20%	20.4%
	18%		18%	
	25%		25%	
	19%		19%	
	20%		20%	
60%	37.5%	33.4%	37.5%	33.4%
	31%		31%	
	28.5%		28.5%	
	41.5%		41.5%	
	28.5%		28.5%	
70%	34.5%	33.4%	34.5%	33.4%
	39.5%		39.5%	
	38%		38%	
	28.5%		28.5%	
	26.5%		26.5%	
80%	58.5%	63.3%	58.5%	63.3%
	56.5%		56.5%	
	61.5%		61.5%	
	70.5%		70.5%	
	69.5%		69.5%	
90%	76.5%	76.1%	76.5%	76.1%
	73.5%		73.5%	
	74.5%		74.5%	
	76.5%		76.5%	
	79.5%		79.5%	
100%	100%	100%	100%	100%

Tabla 3.4 Resultados de Pruebas con enlace **s2** - **s4** sin Pérdidas

% de Pérdidas Configurado en el enlace s1 – s3	% de Pérdidas Obtenido en el enlace s1 – s3 sin Codificación de Red	% de Pérdidas Obtenido en el enlace s1 – s3 con Codificación de Red
10%	4%	7.6%
20%	18%	0%
30%	30.5%	0%
40%	34%	0%
50%	43.5%	0%
60%	48.5%	16.4%
70%	64.5%	0%
80%	78.5%	0%
90%	91.5%	0%
100%	100%	0%

3.2. DISCUSIÓN

En los datos resultantes de las pruebas realizadas se observa una clara disminución del porcentaje de pérdidas en los enlaces al usar codificación de red. La Tabla 3.5 presenta un resumen de los resultados promedio obtenidos sin y con codificación de red bajo distintos porcentajes de pérdidas preestablecidos.

Tabla 3.5 Resumen de Resultados

% de Pérdidas Configurado	% de Pérdidas Obtenido para el enlace s1 – s3 sin Codificación de Red	% de Pérdidas Obtenido para el enlace s2 – s4 sin Codificación de Red	% de Pérdidas Obtenido con Codificación de Red
0%	0%	0%	0%
10%	7.6%	5.9%	0.1%
20%	16.4%	16%	1.3%
30%	25.5%	27.1%	5.3%
40%	37.3%	36.7%	13.7%
50%	46.7%	46.5%	20.4%
60%	61.3%	53.9%	33.4%
70%	67.5%	68.5%	33.4%
80%	74.5%	81.5%	63.3%
90%	87.8%	88.3%	76.1%
100%	100%	100%	100%

Se aprecia que al momento de aplicar codificación de red al sistema el porcentaje de pérdidas disminuye considerablemente, sobre todo en valores cercanos al 50%.

Esto debido a que sin codificación de red cada enlace posee una probabilidad independiente de perder tramas, la cual es prestablecida en el archivo de Python que define la red virtual en Mininet. Sin embargo, al aplicar codificación de red la probabilidad de pérdidas en la red se reduce, ya que existe la posibilidad de recuperar la información a través de las tramas codificadas que viajan por la red.

Solo existe pérdidas de la información si ambos enlaces pierden sus tramas al mismo tiempo, es decir, la probabilidad de que una trama no llegue a su destino deja de ser la probabilidad de pérdidas del enlace y se convierte en la probabilidad de que ambos enlaces pierdan tramas al mismo tiempo.

Esta probabilidad se calcula como la probabilidad de ocurrencia en simultáneo de dos elementos independientes, ya que los enlaces entre **s1** y **s3** y entre **s2** y **s4** no dependen uno de otro en su funcionamiento, por lo tanto, la probabilidad de pérdida en la red *butterfly* con codificación de red se establece como el producto de la probabilidad de pérdida entre el enlace **s1** – **s3** y el enlace **s2** – **s4** como se muestra en la Tabla 3.6.

Tabla 3.6 Comparación Teórico – Práctico de porcentaje de pérdidas

% de Pérdidas sin Codificación de Red (Teórico)	% de Pérdidas con Codificación de Red (Teórico)	% de Pérdidas con Codificación de Red (Obtenido)
0%	0%	0%
10%	1%	0.1%
20%	4%	1.3%
30%	9%	5.3%
40%	16%	13.7%
50%	25%	20.4%
60%	36%	33.4%
70%	49%	33.4%
80%	64%	63.3%
90%	81%	76.1%
100%	100%	100%

Los resultados obtenidos en las pruebas comprueban que los porcentajes de pérdidas obtenidos se acercan a los porcentajes de pérdidas esperados para una red con codificación de red donde la probabilidad de pérdida de la red es igual a la probabilidad de pérdidas simultáneas en ambos enlaces de la red *butterfly*.

Se verifica que el porcentaje de pérdidas se mantiene menor al 50% siempre y cuando los porcentajes de pérdidas en cada uno de los enlaces no superen el 70%, una vez superado este valor las pérdidas de la red con codificación de red se acercan cada vez más a los valores sin codificación de red hasta llegar al límite del

100% de porcentaje de pérdidas en ambos enlaces, donde no se puede recuperar ninguna trama, como se muestra en la Figura 3.1.

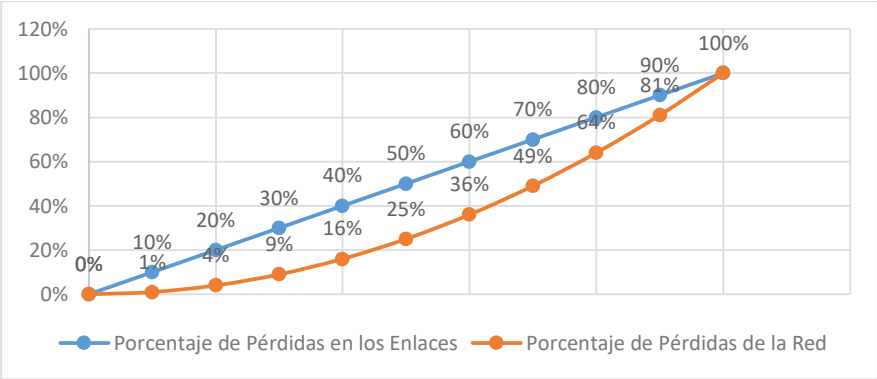


Figura 3.1 Pérdidas Teóricas en Enlaces y la Red

Se puede apreciar como los valores obtenidos en las pruebas realizadas siguen la tendencia de las curvas teóricas en la Figura 3.2, para las pruebas sin codificación de red, y en la Figura 3.3 para las pruebas con codificación de red.

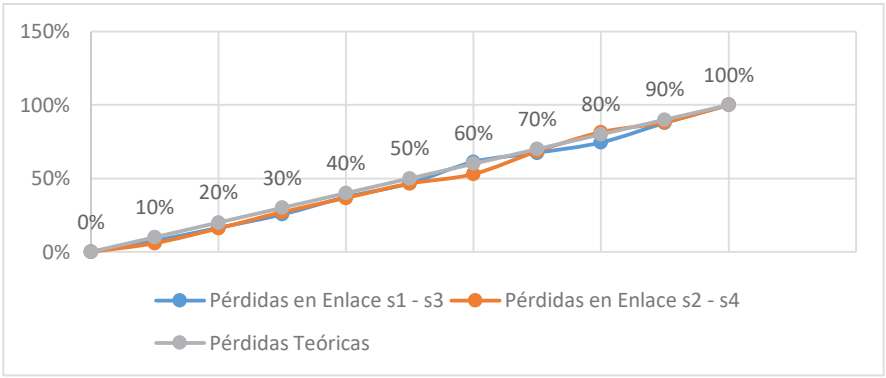


Figura 3.2 Pérdidas sin Codificación de Red

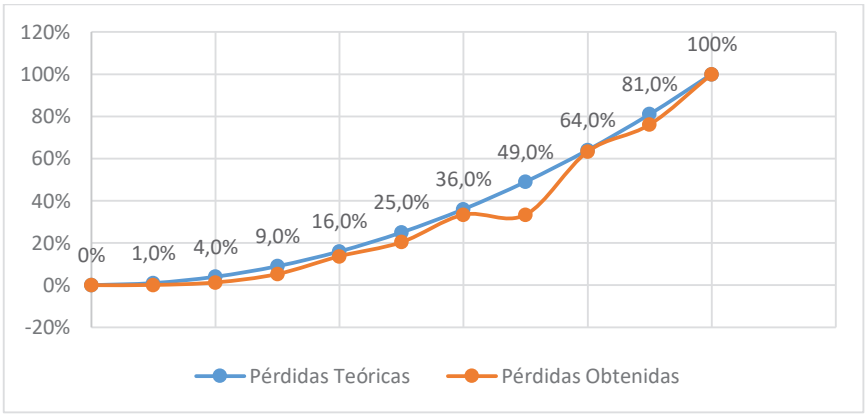


Figura 3.3 Gráfica de Pérdidas con Codificación de Red

En el caso donde únicamente uno de los enlaces posee porcentaje de pérdidas, la codificación de red permite una recuperación total de la información, ya que la existencia garantizada de la trama codificada y una de las tramas de información garantiza a su vez la existencia de la otra trama de información. La probabilidad de que ambos enlaces fallen disminuye considerablemente a casi cero y esto anula la probabilidad de pérdidas configurada en uno de los enlaces.

Finalmente, la Tabla 3.7 presenta el porcentaje de recuperación promedio obtenido para cada prueba, calculado como la diferencia entre el porcentaje de pérdidas con codificación de red y el porcentaje de pérdidas sin codificación de red en cada enlace.

Tabla 3.7 Porcentaje de Recuperación Obtenido

% de Pérdidas Obtenido para el enlace s1 – s3	% de Pérdidas Obtenido para el enlace s2 – s4	% de Pérdidas Obtenido con Codificación de Red	% de Recuperación para el enlace s1 – s3	% de Recuperación para el enlace s2 – s4
0%	0%	0%	0%	0%
7.6%	5.9%	0.1%	7.5%	5.8%
16.4%	16%	1.3%	15.1%	14.7%
25.5%	27.1%	5.3%	20.2%	21.8%
37.3%	36.7%	13.7%	23.6%	23%
46.7%	46.5%	20.4%	26.3%	26.1%
61.3%	53.9%	33.4%	27.9%	20.5%
67.5%	68.5%	33.4%	34.1%	25.1%
74.5%	81.5%	63.3%	11.2%	18.2%
87.8%	88.3%	76.1%	11.7%	12.2%
100%	100%	100%	0%	0%

CAPÍTULO 4. CONCLUSIONES Y RECOMENDACIONES

Las aplicaciones desarrolladas para implementar los mecanismos de codificación y decodificación de tramas son indispensables para la implementación de esta variante de codificación de red, no solamente porque las reglas que instala el controlador requieren el reenvío de toda trama a estas aplicaciones, sino por la limitación del protocolo OpenFlow 1.3 para modificar los contenidos de una trama de datos más allá de las cabeceras de los protocolos de red de capa 2, 3 o 4.

La codificación de red genera retardos en la transmisión de datos debido al procesamiento realizado en cada uno de los nodos de la red. A esto se añade el tiempo que le toma a la trama recorrer los nodos necesarios para llegar al equipo codificador y el tiempo de espera para generar una trama codificada con otra trama de datos. Estos retardos deben ser cuidadosamente revisados con el fin de no generar la impresión errónea que la trama se ha perdido durante la transmisión a los equipos terminales.

La creación de tramas codificadas no se puede realizar en el controlador SDN debido a que OpenFlow 1.3 no envía la trama de datos original al controlador cuando una trama ingresa, en su lugar manda información de la trama que ingresa junto con una copia de los datos, la trama original se almacena en *buffers* del dispositivo de red y es alterada o reenviada solo por las reglas establecidas por el controlador. Es decir, el controlador SDN no puede crear o editar el contenido de las tramas de manera directa, sino que lo debe hacer a través de reglas instaladas en los dispositivos de red.

Las pruebas realizadas utilizan ICMP debido a su facilidad de seguimiento e interpretación de resultados. ICMP no realiza reenvío en caso de no recibir una respuesta, y esto facilita la determinación del número de tramas perdidas y el número de tramas recuperadas gracias a la codificación de red. El mecanismo de

petición y respuesta individual del comando `ping` permite realizar un seguimiento paso a paso del trayecto de cada trama a través de la red con topología *butterfly*, lo que facilitó la obtención de resultados.

En base a los resultados obtenidos, se comprueba que las aplicaciones realizadas permiten la utilización de la codificación de red como una herramienta de recuperación de errores, donde las tramas perdidas en uno de los enlaces de la red *butterfly* pueden ser recuperados en base a tramas adicionales enviadas a través de la red y este procedimiento es transparente para los equipos terminales.

Se comprueba que la codificación de red permite la reducción de la probabilidad de pérdidas en una red *butterfly*. No se puede anular totalmente las pérdidas de información debido a que dentro de una red *butterfly* con dos enlaces con pérdidas, si bien la probabilidad de perder una trama es independiente para cada enlace, existe la probabilidad de que ambos enlaces pierdan una trama al mismo tiempo, lo cual imposibilita la recuperación de información.

A medida que el porcentaje de pérdidas en cada enlace aumenta, la codificación de red se vuelve menos eficiente en recuperar tramas perdidas, debido a que aumenta la probabilidad de dos pérdidas simultáneas en la red.

Se considera que el mecanismo de codificación de red para recuperación de tramas perdidas es una herramienta viable para valores de pérdidas en los enlaces variables. La respuesta de la aplicación a distintos porcentajes de pérdidas depende de la probabilidad de pérdida simultánea de la red.

La red implementada genera retardos debido a la necesidad de controlar el orden en que las tramas ingresarán a los distintos nodos. Sin embargo, estos retardos se compensan con una mejora de la confiabilidad de la red, reduciendo la necesidad de retransmisión de información.

La simulación de envío simultáneo de tramas con el comando `at` permitió enviar tramas con la menor diferencia de tiempo posible. Estas diferencias de tiempo entre tramas se deben a los relojes internos de los procesadores de cada terminal y que no pueden ser totalmente sincronizados por un servidor NTP debido a retardos, así como el resto de aplicaciones que se encuentran ejecutando en cada uno de los terminales y que ocupan espacios de tiempo en el procesador que no pueden ser controlados debido a que no se puede determinar cuándo se asignan tiempos en el procesador.

Si bien un escenario real puede no presentar el envío de dos tramas simultáneamente desde dos terminales, se puede considerar la implementación de la codificación de red en una red de core o una red con tráfico constante de datos.

Se recomienda mejorar el tiempo de respuesta de las aplicaciones codificadoras y decodificadoras de red utilizando equipos con mayor capacidad de procesamiento y menos aplicaciones secundarias ejecutándose dentro del sistema operativo, con el fin de disminuir los retardos producidos por el proceso de codificación de red en la aplicación.

Se recomienda realizar pruebas de codificación de red y compararlo con aplicaciones que usen TCP con el fin de determinar si existe una mejora en la ventana de envío de TCP, lo cual reflejaría la mejora en la confiabilidad de la red debido a la reducción de retransmisiones de paquetes.

El ambiente virtual desarrollado en Mininet permitió un alto rango de pruebas ante diferentes porcentajes de pérdidas, algo mucho más difícil de lograr utilizando equipos físicos. Sin embargo, la utilización de 7 máquinas virtuales dentro de una misma máquina fisica representa una gran utilización de recursos y por lo tanto se recomienda ejecutar el presente proyecto en dos o más equipos para un mejor desempeño.

El presente trabajo puede ser posteriormente desarrollado con el fin de implementar la codificación de red en grupos multicast, donde la red pueda recuperar tramas perdidas por un grupo de usuarios.

Se puede probar el desempeño de la aplicación en redes inalámbricas, con una implementación en equipos físicos que permita analizar el comportamiento de esta solución en un ambiente donde la transmisión de tramas entre **s1** y **s3** y entre **s2** y **s4** se realiza de manera inalámbrica.

Se recomienda utilizar una nueva versión del protocolo OpenFlow que permita deshabilitar la utilización de *buffers* en los *switches* o crear nuevas tramas dentro de la aplicación que se ejecuta sobre el controlador.

Se recomienda instalar las aplicaciones de codificación y decodificación de tramas en equipos con características de hardware bastante altas, esto con el propósito de disminuir los tiempos de procesamiento de tramas y disminuir el tiempo total de transmisión de datos.

REFERENCIAS BIBLIOGRÁFICAS

[1] W. Braun y M. Menth, «Software-Defined Networking Using OpenFlow: Protocols, Applications and Architectural Design Choices,» *Future Internet,* 2014.

[2] P. B. G. Y. W. M. O. W. y. K. R. Alexandros G. Dimakis, «Network Coding for Distributed Storage Systems,» California, Berkeley, 2008.

[3] Open Networking Foundation, «OpenFlow Switch Specification,» Open Networking Foundation, 2012.

[4] C. Fernandez y J. Muñoz, Software Defined Networking (SDN) with OpenFlow 1.3, Open vSwitch and Ryu, Universidad Politécnica de Cataluña: UPC Telematics Department, 2011.

[5] Ryu SDN Framework Community, «Ryu SDN Framework,» ChocoTemplates, 2014. [En línea]. Available: https://osrg.github.io/ryu/. [Último acceso: Abril 2016].

[6] Mininet Team , «Mininet,» Octopress , 2016. [En línea]. Available: http://mininet.org/download/. [Último acceso: Marzo 2016].

[7] T. Ho y D. Lun, Network Coding: An Introduction, 2015.

[8] Mininet Team , «Download/Get Started With Mininet,» Mininet Team , 2017. [En línea]. Available: http://mininet.org/download/. [Último acceso: 1 05 2016].

[9] Nippon Telegraph and Telephone Corporation, «Getting Started Ryu,» Nippon Telegraph and Telephone Corporation, 2017. [En línea]. Available: http://ryu.readthedocs.io/en/latest/getting_started.html. [Último acceso: 1 05 2016].

[10] Open Networking Foundation, «Software-Defined Networking: The New Norm for Networks,» Open Networking Foundation, 2012.

ANEXOS

ANEXO I: CAMPOS DE MATCH ESPECIFICADOS EN OPENFLOW 1.3

ANEXO II: CODIGO DEL AMBIENTE DE RED DE MININET

ANEXO III: CODIGO DE LA APLICACIÓN PARA EL CONTROLADOR SIN CODIFICACIÓN DE RED

ANEXO IV: CODIGO DE LA APLICACIÓN PARA EL CONTROLADOR CON CODIFICACIÓN DE RED

ANEXO V: APLICACIÓN PARA CODIFICACIÓN DE RED EN NC SWITCH5

ANEXO VI: APLICACIÓN PARA CODIFICACIÓN DE RED EN NC SWITCH6

ANEXO VII: MANUAL DE USUARIO

ANEXO I:

CAMPOS DE MATCH ESPECIFICADOS EN OPENFLOW 1.3

```
/* OXM Flow match field types for OpenFlow basic class. */
enum oxm_ofb_match_fields {
    OFPXMT_OFB_IN_PORT        = 0,  /* Switch input port. */
    OFPXMT_OFB_IN_PHY_PORT    = 1,  /* Switch physical input port. */
    OFPXMT_OFB_METADATA       = 2,  /* Metadata passed between tables. */
    OFPXMT_OFB_ETH_DST        = 3,  /* Ethernet destination address. */
    OFPXMT_OFB_ETH_SRC        = 4,  /* Ethernet source address. */
    OFPXMT_OFB_ETH_TYPE       = 5,  /* Ethernet frame type. */
    OFPXMT_OFB_VLAN_VID       = 6,  /* VLAN id. */
    OFPXMT_OFB_VLAN_PCP       = 7,  /* VLAN priority. */
    OFPXMT_OFB_IP_DSCP        = 8,  /* IP DSCP (6 bits in ToS field). */
    OFPXMT_OFB_IP_ECN         = 9,  /* IP ECN (2 bits in ToS field). */
    OFPXMT_OFB_IP_PROTO       = 10, /* IP protocol. */
    OFPXMT_OFB_IPV4_SRC       = 11, /* IPv4 source address. */
    OFPXMT_OFB_IPV4_DST       = 12, /* IPv4 destination address. */
    OFPXMT_OFB_TCP_SRC        = 13, /* TCP source port. */
    OFPXMT_OFB_TCP_DST        = 14, /* TCP destination port. */
    OFPXMT_OFB_UDP_SRC        = 15, /* UDP source port. */
    OFPXMT_OFB_UDP_DST        = 16, /* UDP destination port. */
    OFPXMT_OFB_SCTP_SRC       = 17, /* SCTP source port. */
    OFPXMT_OFB_SCTP_DST       = 18, /* SCTP destination port. */
    OFPXMT_OFB_ICMPV4_TYPE    = 19, /* ICMP type. */
    OFPXMT_OFB_ICMPV4_CODE    = 20, /* ICMP code. */
    OFPXMT_OFB_ARP_OP         = 21, /* ARP opcode. */
    OFPXMT_OFB_ARP_SPA        = 22, /* ARP source IPv4 address. */
    OFPXMT_OFB_ARP_TPA        = 23, /* ARP target IPv4 address. */
    OFPXMT_OFB_ARP_SHA        = 24, /* ARP source hardware address. */
    OFPXMT_OFB_ARP_THA        = 25, /* ARP target hardware address. */
    OFPXMT_OFB_IPV6_SRC       = 26, /* IPv6 source address. */
    OFPXMT_OFB_IPV6_DST       = 27, /* IPv6 destination address. */
    OFPXMT_OFB_IPV6_FLABEL    = 28, /* IPv6 Flow Label */
    OFPXMT_OFB_ICMPV6_TYPE    = 29, /* ICMPv6 type. */
    OFPXMT_OFB_ICMPV6_CODE    = 30, /* ICMPv6 code. */
    OFPXMT_OFB_IPV6_ND_TARGET = 31, /* Target address for ND. */
    OFPXMT_OFB_IPV6_ND_SLL    = 32, /* Source link-layer for ND. */
    OFPXMT_OFB_IPV6_ND_TLL    = 33. /* Target link-layer for ND. */
    OFPXMT_OFB_MPLS_LABEL     = 34, /* MPLS label. */
    OFPXMT_OFB_MPLS_TC        = 35, /* MPLS TC. */
    OFPXMT_OFP_MPLS_BOS       = 36, /* MPLS BoS bit. */
    OFPXMT_OFB_PBB_ISID       = 37, /* PBB I-SID. */
    OFPXMT_OFB_TUNNEL_ID      = 38, /* Logical Port Metadata. */
    OFPXMT_OFB_IPV6_EXTHDR    = 39, /* IPv6 Extension Header pseudo-field */
};
```

ANEXO II:
CÓDIGO DEL AMBIENTE DE RED DE MININET

```
#!/usr/bin/python

from mininet.cli import CLI
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.log import setLogLevel, info
from mininet.node import RemoteController
from mininet.link import TCLink
from mininet.link import Intf
from ConfigParser import ConfigParser

#abrir archivo de configuracion
cfg = ConfigParser()
archivo = "Perdidas.ini"
cfg.read(archivo)
#Importar perdidas del archivo de configuracion
PERDIDAS_S1_S3 = int(cfg.get('perdidas','perdidas_s1_s3'))
PERDIDAS_S2_S4 = int(cfg.get('perdidas','perdidas_s2_s4'))

#Clase para Toplogía Butterfly
class Butterfly( Topo ):
    def __init__( self ):
        # Inicializar la Topologia
        Topo.__init__( self )

        # Crear Switches
        S1 = self.addSwitch('s1')
        S2 = self.addSwitch('s2')
        S3 = self.addSwitch('s3')
        S4 = self.addSwitch('s4')
        S5 = self.addSwitch('s5')
        S6 = self.addSwitch('s6')

        self.addLink(S1, S3, port1=2, port2=2, bw=100, delay='10ms', loss=PERDIDAS_S1_S3)
        self.addLink(S2, S4, port1=2, port2=2, bw=100, delay='10ms', loss=PERDIDAS_S2_S4)
        self.addLink(S1, S5, port1=3, port2=1, bw=100, delay='1ms')
        self.addLink(S2, S5, port1=3, port2=2, bw=100, delay='1ms')
        self.addLink(S3, S6, port1=3, port2=1, bw=100, delay='1ms')
        self.addLink(S4, S6, port1=3, port2=2, bw=100, delay='1ms')
        self.addLink(S5, S6, port1=3, port2=3, bw=100, delay='20ms')

if __name__ == '__main__':
    #Establecer nivel de logs en información
    setLogLevel('info')

    #Crear instancia de Mininet
    net = Mininet(topo=Butterfly(), controller=RemoteController('c0', ip='127.0.0.1'),
link=TCLink)

    #Asociación de interfaces de red con puertos de los switches virtuales
    Intf('ens33', net.switches[0], port=1)
    info('*** Adding hardware interface', 'ens33', 'to switch', net.switches[0].name, '\n')
    Intf('ens34', net.switches[1], port=1)
    info('*** Adding hardware interface', 'ens34', 'to switch', net.switches[1].name, '\n')
    Intf('ens35', net.switches[2], port=1)
    info('*** Adding hardware interface', 'ens35', 'to switch', net.switches[2].name, '\n')
    Intf('ens36', net.switches[3], port=1)
    info('*** Adding hardware interface', 'ens36', 'to switch', net.switches[3].name, '\n')
    Intf('ens37', net.switches[4], port=4)
    info('*** Adding hardware interface', 'ens36', 'to switch', net.switches[4].name, '\n')
    Intf('ens38', net.switches[5], port=4)
    info('*** Adding hardware interface', 'ens36', 'to switch', net.switches[5].name, '\n')
    #Inicio, ejecución y finalización del ambiente virtual de red
    net.start()
    CLI(net)
    net.stop()
```

ANEXO III:

CÓDIGO DE LA APLICACIÓN PARA EL CONTROLADOR SIN CODIFICACIÓN DE RED

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ConfigParser import ConfigParser

cfg = ConfigParser()
archivo ="MACaddressRYU.ini"
cfg.read(archivo)

#Importar direcciones MAC del archivo de configuracion
MAC_HOST1 = cfg.get('macs_sinNC','mac_host1')
MAC_HOST2 = cfg.get('macs_sinNC','mac_host2')
MAC_HOST3 = cfg.get('macs_sinNC','mac_host3')
MAC_HOST4 = cfg.get('macs_sinNC','mac_host4')

#Aplicación sin Network Coding
class SwitchNC(app_manager.RyuApp):

    # Especificar la version del protocolo OpenFlow 1.3
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    # Inicializar la aplicación
    def __init__(self, *args, **kwargs):
        super(SwitchNC, self).__init__(*args, **kwargs) \

        # Contador de Switches configurados
        self.id_sw = 0

    # Método que se ejecuta al momento de generarse el evento EventOFPSwitchFeatures
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):

        #Obtener la informacion del Switch
        datapath = ev.msg.datapath

        #Reglas para s1
        if datapath.id == 1:

            self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                         in_port=1, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                         in_port=2, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.id_sw += 1

        if datapath.id == 2:

            self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                         in_port=1, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                         in_port=2, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)
```

```
            self.id_sw += 1

        if datapath.id == 3:

            self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                         in_port=1, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                         in_port=2, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.id_sw += 1

        if datapath.id == 4:

            self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                         in_port=1, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                         in_port=2, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.id_sw += 1

        if self.id_sw == 4:

            self.logger.info("--------------RED CONFIGURADA CORRECTAMENTE-------------")
            self.logger.info("--------------------------------------------------------")
            self.logger.info("|           ||         MATCH      ||      ACTIONS     ||")
            self.logger.info("| DATAPATH  ||-----------------------------------------")
            self.logger.info("|           || INPORT | SRC | DST || MOD_DST | OUTPUT ||")
            self.logger.info("--------------------------------------------------------")
            self.logger.info("|     1     ||    1   | :11 | :33 ||   NONE  |    2   ||")
            self.logger.info("|     1     ||    2   | :33 | :11 ||   NONE  |    1   ||")
            self.logger.info("--------------------------------------------------------")
            self.logger.info("|     2     ||    1   | :22 | :44 ||   NONE  |    2   ||")
            self.logger.info("|     2     ||    2   | :44 | :22 ||   NONE  |    1   ||")
            self.logger.info("--------------------------------------------------------")
            self.logger.info("|     3     ||    1   | :33 | :11 ||   NONE  |    2   ||")
            self.logger.info("|     3     ||    2   | :11 | :33 ||   NONE  |    1   ||")
            self.logger.info("--------------------------------------------------------")
            self.logger.info("|     4     ||    1   | :44 | :22 ||   NONE  |    2   ||")
            self.logger.info("|     4     ||    2   | :22 | :44 ||   NONE  |    1   ||")
            self.logger.info("--------------------------------------------------------")

            self.id_sw += 1

    # Instalar las reglas en los Switches
    def addFlow(self, priority, eth_src, eth_dst, in_port, out_port, parser, datapath,
ofproto):

        match = parser.OFPMatch(in_port=in_port, eth_src=eth_src, eth_dst=eth_dst)
        actions = [parser.OFPActionOutput(out_port)]
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match,
instructions=inst)
        datapath.send_msg(mod)
```

ANEXO IV:

CÓDIGO DE LA APLICACIÓN PARA EL CONTROLADOR CON CODIFICACIÓN DE RED

```
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ConfigParser import ConfigParser

cfg = ConfigParser()
archivo = "MACaddressRYU.ini"
cfg.read(archivo)

#Importar direcciones MAC del archivo de configuracion
MAC_HOST1 = cfg.get('macs_conNC','mac_host1')
MAC_HOST2 = cfg.get('macs_conNC','mac_host2')
MAC_HOST3 = cfg.get('macs_conNC','mac_host3')
MAC_HOST4 = cfg.get('macs_conNC','mac_host4')
MAC_SWITCH5 = cfg.get('macs_conNC','mac_switch5')
MAC_SWITCH6 = cfg.get('macs_conNC','mac_switch6')
MAC_HOST1_U = cfg.get('macs_conNC','mac_host1_u')
MAC_HOST2_U = cfg.get('macs_conNC','mac_host2_u')
MAC_HOST3_U = cfg.get('macs','mac_host3_u')
MAC_HOST4_U = cfg.get('macs_conNC','mac_host4_u')
MAC_H3XORH4_77 = cfg.get('macs_conNC','mac_h3xorh4_77')
MAC_H1XORH2_88 = cfg.get('macs_conNC','mac_h1xorh2_88')

#Controlador para Network Coding
class SwitchNC(app_manager.RyuApp):

    # Especificar la version del protocolo OpenFlow 1.3
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    # Inicializar el controlador
    def __init__(self, *args, **kwargs):
        super(SwitchNC, self).__init__(*args, **kwargs) \

        # Contador de Switches configurados
        self.id_sw = 0

    # Etapa de configuracion
    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
    def switch_features_handler(self, ev):

        #Obtener la informacion del Switch
        datapath = ev.msg.datapath

        # Reglas para s1
        if datapath.id == 1:

            self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                         in_port=1, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                         in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                         in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                         datapath=ev.msg.datapath,
                         ofproto=ev.msg.datapath.ofproto)

            self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
```

```
                in_port=3, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST1_U, eth_dst=MAC_HOST3,
                in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3_U, eth_dst=MAC_HOST1,
                in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.id_sw += 1

    if datapath.id == 2:

        self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                in_port=1, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                in_port=3, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2_U, eth_dst=MAC_HOST4,
                in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4_U, eth_dst=MAC_HOST2,
                in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.id_sw += 1

    if datapath.id == 3:

        self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                in_port=1, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                in_port=3, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3_U, eth_dst=MAC_HOST1,
                in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
```

```
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST1_U, eth_dst=MAC_HOST3,
                in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.id_sw += 1

    if datapath.id == 4:

        self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                in_port=1, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                in_port=3, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4_U, eth_dst=MAC_HOST2,
                in_port=3, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2_U, eth_dst=MAC_HOST4,
                in_port=2, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.id_sw += 1

    if datapath.id == 5:

        self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                in_port=1, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                in_port=2, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                in_port=1, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                in_port=2, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_H3XORH4_77, eth_dst=MAC_H1XORH2_88,
                in_port=3, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                in_port=4, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
```

```
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                in_port=4, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                in_port=4, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                in_port=4, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_H1XORH2_88, eth_dst=MAC_H3XORH4_77,
                in_port=4, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST1_U, eth_dst=MAC_HOST3,
                in_port=4, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2_U, eth_dst=MAC_HOST4,
                in_port=4, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3_U, eth_dst=MAC_HOST1,
                in_port=1, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4_U, eth_dst=MAC_HOST2,
                in_port=2, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)



        self.id_sw += 1

    if datapath.id == 6:

        self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
                in_port=1, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                in_port=2, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                in_port=1, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                in_port=2, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_H1XORH2_88, eth_dst=MAC_H3XORH4_77,
                in_port=3, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST1, eth_dst=MAC_HOST3,
```

```
                in_port=4, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2, eth_dst=MAC_HOST4,
                in_port=4, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3, eth_dst=MAC_HOST1,
                in_port=4, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4, eth_dst=MAC_HOST2,
                in_port=4, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_H3XORH4_77, eth_dst=MAC_H1XORH2_88,
                in_port=4, out_port=3, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST1_U, eth_dst=MAC_HOST3,
                in_port=1, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST2_U, eth_dst=MAC_HOST4,
                in_port=2, out_port=4, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST3_U, eth_dst=MAC_HOST1,
                in_port=4, out_port=1, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.addFlow(priority=0, eth_src=MAC_HOST4_U, eth_dst=MAC_HOST2,
                in_port=4, out_port=2, parser=ev.msg.datapath.ofproto_parser,
                datapath=ev.msg.datapath,
                ofproto=ev.msg.datapath.ofproto)

        self.id_sw += 1

    if self.id_sw == 6:

        self.logger.info('--------------RED CONFIGURADA CORRECTAMENTE-------------')
        self.logger.info('---------------------------------------------------------')
        self.logger.info('|           ||         MATCH       ||      ACTIONS     ||')
        self.logger.info('| DATAPATH  ||-------------------------------------------')
        self.logger.info('|           || INPORT | SRC | DST || MOD_DST | OUTPUT ||')
        self.logger.info('---------------------------------------------------------')
        self.logger.info('|     1     ||    1   | :11 | :33 ||   NONE  |    3   ||')
        self.logger.info('|     1     ||    2   | :33 | :11 ||   NONE  |    3   ||')
        self.logger.info('|     1     ||    3   | :11 | :33 ||   NONE  |    2   ||')
        self.logger.info('|     1     ||    3   | :33 | :11 ||   NONE  |    1   ||')
        self.logger.info('---------------------------------------------------------')
        self.logger.info('|     2     ||    1   | :22 | :44 ||   NONE  |    3   ||')
        self.logger.info('|     2     ||    2   | :44 | :22 ||   NONE  |    3   ||')
        self.logger.info('|     2     ||    3   | :22 | :44 ||   NONE  |    2   ||')
        self.logger.info('|     2     ||    3   | :44 | :22 ||   NONE  |    1   ||')
        self.logger.info('---------------------------------------------------------')
        self.logger.info('|     3     ||    1   | :33 | :11 ||   NONE  |    3   ||')
        self.logger.info('|     3     ||    2   | :11 | :33 ||   NONE  |    3   ||')
        self.logger.info('|     3     ||    3   | :33 | :11 ||   NONE  |    2   ||')
        self.logger.info('|     3     ||    3   | :11 | :33 ||   NONE  |    1   ||')
        self.logger.info('---------------------------------------------------------')
        self.logger.info('|     4     ||    1   | :44 | :22 ||   NONE  |    3   ||')
        self.logger.info('|     4     ||    2   | :22 | :44 ||   NONE  |    3   ||')
        self.logger.info('|     4     ||    3   | :44 | :22 ||   NONE  |    2   ||')
        self.logger.info('|     4     ||    3   | :22 | :44 ||   NONE  |    1   ||')
        self.logger.info('---------------------------------------------------------')
```

```
            self.logger.info('|     5     ||    1    | :11 | :33 ||    :55    |    4    ||')
            self.logger.info('|     5     ||    2    | :22 | :44 ||    :55    |    4    ||')
            self.logger.info('|     5     ||    1    | :33 | :11 ||    :55    |    4    ||')
            self.logger.info('|     5     ||    2    | :44 | :22 ||    :55    |    4    ||')
            self.logger.info('|     5     ||    3    | :77 | :88 ||    :55    |    4    ||')
            self.logger.info('|     5     ||    4    | :11 | :33 ||   NONE    |    1    ||')
            self.logger.info('|     5     ||    4    | :22 | :44 ||   NONE    |    2    ||')
            self.logger.info('|     5     ||    4    | :33 | :11 ||   NONE    |    1    ||')
            self.logger.info('|     5     ||    4    | :44 | :22 ||   NONE    |    2    ||')
            self.logger.info('|     5     ||    4    | :88 | :77 ||   NONE    |    3    ||')
            self.logger.info('-------------------------------------------------------------')
            self.logger.info('|     6     ||    1    | :11 | :33 ||    :66    |    4    ||')
            self.logger.info('|     6     ||    2    | :22 | :44 ||    :66    |    4    ||')
            self.logger.info('|     6     ||    1    | :33 | :11 ||    :66    |    4    ||')
            self.logger.info('|     6     ||    2    | :44 | :22 ||    :66    |    4    ||')
            self.logger.info('|     6     ||    3    | :88 | :77 ||    :66    |    4    ||')
            self.logger.info('|     6     ||    4    | :11 | :33 ||   NONE    |    1    ||')
            self.logger.info('|     6     ||    4    | :22 | :44 ||   NONE    |    2    ||')
            self.logger.info('|     6     ||    4    | :33 | :11 ||   NONE    |    1    ||')
            self.logger.info('|     6     ||    4    | :44 | :22 ||   NONE    |    2    ||')
            self.logger.info('|     6     ||    4    | :77 | :88 ||   NONE    |    3    ||')
            self.logger.info('-------------------------------------------------------------')

    # Instalar las reglas en los Switches
    def addFlow(self, priority, eth_src, eth_dst, in_port, out_port, parser, datapath,
ofproto):

        #Regla a instalar en s1, s2, s3 o s4
        if datapath.id in [1, 2, 3, 4]:
            actions = [parser.OFPActionOutput(out_port)]

        # Regla a instalar en s5 o s6
        if datapath.id in [5, 6]:

            # Regla si s5 o s6 envia a NC Switch5 o NC Switch6
            if out_port == 4:

                if datapath.id == 5:
                    new_mac_dst = MAC_SWITCH5

                if datapath.id == 6:
                    new_mac_dst = MAC_SWITCH6

                actions = [parser.OFPActionSetField(eth_dst=new_mac_dst),
parser.OFPActionOutput(out_port)]

            # Regla si s5 o s6 recibe desde NC Switch5 o NC Switch6
            if in_port == 4:
                actions = [parser.OFPActionOutput(out_port)]

        match = parser.OFPMatch(in_port=in_port, eth_src=eth_src, eth_dst=eth_dst)
        inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS, actions)]
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority, match=match,
instructions=inst)
        datapath.send_msg(mod)
```

ANEXO V:

CÓDIGO DE LA APLICACIÓN PARA CODIFICACIÓN DE RED EN NC SWITCH5

```
import socket
import os
import threading
from time import clock
import ConfigParser

cfg = ConfigParser.ConfigParser()
archivo ="MACaddress.ini"
cfg.read(archivo)

#Importar direcciones MAC del archivo de configuracion
MAC_HOST1 = cfg.get('macs','mac_host1')
MAC_HOST2 = cfg.get('macs','mac_host2')
MAC_HOST3 = cfg.get('macs','mac_host3')
MAC_HOST4 = cfg.get('macs','mac_host4')
MAC_SWITCH5 = cfg.get('macs','mac_switch5')
MAC_SWITCH6 = cfg.get('macs','mac_switch6')
MAC_HOST1_U = cfg.get('macs','mac_host1_u')
MAC_HOST2_U = cfg.get('macs','mac_host2_u')
MAC_HOST3_U = cfg.get('macs','mac_host3_u')
MAC_HOST4_U = cfg.get('macs','mac_host4_u')
MAC_SWITCH5_U = cfg.get('macs','mac_switch5_u')
MAC_SWITCH6_U = cfg.get('macs','mac_switch6_u')
MAC_H3XORH4_77 = cfg.get('macs','mac_h3xorh4_77')
MAC_H1XORH2_88 = cfg.get('macs','mac_h1xorh2_88')

# Creacion del socket para capturar tramas
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0003))
s.bind(('ens33', 0))

print '\nSocket Creado Correctamente...'
print '\nEscuchando por Paquetes...'

# Vectores para tramas de cada VM
nchost1_frames = {}
nchost2_frames = {}
nchost3_frames = {}
nchost4_frames = {}
ncswitch5_frames = {}
ncswitch6_frames = {}

#Contador de tramas de cada VM
frame_nchost1 = 0
frame_nchost2 = 0
frame_nchost3 = 0
frame_nchost4 = 0
frame_ncswitch5 = 0
frame_ncswitch6 = 0

#Contadores para NC:
frame_recuperados_total = 0
frame_perdidos = 0
frames_recuperados_nchost3 = 0
frames_recuperados_nchost4 = 0

recuperacion = 0

#Contadores para tramas sin NC
nchost1_noNC = 0
nchost2_noNC = 0
nchost3_noNC = 0
nchost4_noNC = 0

espera = False
```

```
noNCframes = 0
hilorun = 0
tiempo = 0

#Variable para reloj
reloj = clock()

#Temporizados para enviar paquetes sobrantes
def sinNC():

    global hilorun
    global tiempo
    global noNCframes
    global nchost1_noNC
    global nchost2_noNC
    global nchost3_noNC
    global nchost4_noNC
    global frame_nchost1
    global frame_nchost2
    global frame_nchost3
    global frame_nchost4
    global frame_ncswitch5
    global frame_ncswitch6
    global reloj
    global espera

    hilorun = 1
    x = 1

    while True:

        reloj = clock()

        while espera:
            tiempo = 1
            if clock() - reloj > 1 and espera:
                if frame_nchost1 > frame_nchost2 == frame_ncswitch5 and espera and clock()
- reloj > 1:

                    trama_sw1 = nchost1_frames[frame_nchost1]
                    trama_sw1 = trama_sw1[0:12] + MAC_HOST1_U + trama_sw1[24:]
                    s.send(trama_sw1.decode('hex'))

                    frame_nchost2 += 1
                    frame_ncswitch5 += 1
                    noNCframes += 1
                    nchost1_noNC += 1
                    if frame_nchost1 == frame_nchost2:
                        espera = False
                        tiempo = 0
                if frame_nchost2 > frame_nchost1 == frame_ncswitch5 and espera and clock()
- reloj > x:

                    trama_sw2 = nchost2_frames[frame_nchost2]
                    trama_sw2 = trama_sw2[0:12] + MAC_HOST2_U + trama_sw2[24:]
                    s.send(trama_sw2.decode('hex'))

                    frame_nchost1 += 1
                    frame_ncswitch5 += 1
                    noNCframes += 1
                    nchost2_noNC += 1
                    if frame_nchost2 == frame_nchost1:
                        espera = False
                        tiempo = 0
                reloj = clock()


def vista():
    global frames_recuperados_nchost3
    global frames_recuperados_nchost4
    global frame_perdidos
    global recuperacion
    global noNCframes
    global hilorun
```

```
global tiempo
global frame_nchost1
global frame_nchost2
global frame_nchost3
global frame_nchost4
global frame_ncswitch5
global frame_ncswitch6

frames_recuperados_nchost3_1 = frames_recuperados_nchost3
frames_recuperados_nchost4_1 = frames_recuperados_nchost4
frame_perdidos_1 = frame_perdidos
recuperacion_1 = recuperacion
noNCframes_1 = noNCframes
hilorun_1 = hilorun
tiempo_1 = tiempo
frame_nchost1_1 = frame_nchost1
frame_nchost2_1 = frame_nchost2
frame_nchost3_1 = frame_nchost3
frame_nchost4_1 = frame_nchost4
frame_ncswitch5_1 = frame_ncswitch5
frame_ncswitch6_1 = frame_ncswitch6

os.system('clear')
print '\nEstadisticas de Paquetes: \n'
print 'frame_nchost1: {0}'.format(frame_nchost1_1)
print 'frame_nchost2: {0}'.format(frame_nchost2_1)
print 'frame_nchost3: {0}'.format(frame_nchost3_1)
print 'frame_nchost4: {0}'.format(frame_nchost4_1)
print 'frame_ncswitch5: {0}'.format(frame_ncswitch5_1)
print 'frame_ncswitch6: {0}'.format(frame_ncswitch6_1)
print 'frames_recuperados_nchost3 : {0}'.format(frames_recuperados_nchost3_1)
print 'frames_recuperados_nchost4 : {0}'.format(frames_recuperados_nchost4_1)
print 'frame_perdidos       : {0}'.format(frame_perdidos_1)

while True:

    if frames_recuperados_nchost3_1 != frames_recuperados_nchost3 or \
                    frames_recuperados_nchost4_1 != frames_recuperados_nchost4 or \
                    frame_perdidos_1 != frame_perdidos or \
                    recuperacion_1 != recuperacion or \
                    noNCframes_1 != noNCframes or \
                    hilorun_1 != hilorun or \
                    tiempo_1 != tiempo or \
                    frame_nchost1_1 != frame_nchost1 or \
                    frame_nchost2_1 != frame_nchost2 or \
                    frame_nchost3_1 != frame_nchost3 or \
                    frame_nchost4_1 != frame_nchost4 or \
                    frame_ncswitch5_1 != frame_ncswitch5 or \
                    frame_ncswitch6_1 != frame_ncswitch6:

        frames_recuperados_nchost3_1 = frames_recuperados_nchost3
        frames_recuperados_nchost4_1 = frames_recuperados_nchost4
        frame_perdidos_1 = frame_perdidos
        recuperacion_1 = recuperacion
        noNCframes_1 = noNCframes
        hilorun_1 = hilorun
        tiempo_1 = tiempo
        frame_nchost1_1 = frame_nchost1
        frame_nchost2_1 = frame_nchost2
        frame_nchost3_1 = frame_nchost3
        frame_nchost4_1 = frame_nchost4
        frame_ncswitch5_1 = frame_ncswitch5
        frame_ncswitch6_1 = frame_ncswitch6

        os.system('clear')
        print '\nEstadisticas de Paquetes: \n'
        print 'frame_nchost1: {0}'.format(frame_nchost1_1)
        print 'frame_nchost2: {0}'.format(frame_nchost2_1)
        print 'frame_nchost3: {0}'.format(frame_nchost3_1)
        print 'frame_nchost4: {0}'.format(frame_nchost4_1)
        print 'frame_ncswitch5: {0}'.format(frame_ncswitch5_1)
        print 'frame_ncswitch6: {0}'.format(frame_ncswitch6_1)
        print 'Paquetes RecuperadosSW3 : {0}'.format(frames_recuperados_nchost3_1)
        print 'Paquetes RecuperadosSW4 : {0}'.format(frames_recuperados_nchost4_1)
```

```
            print 'Paquetes Perdidos       : {0}'.format(frame_perdidos_l)
            print 'Recuperacion            : {0}'.format(recuperacion_l)
            print 'Tramas enviadas sin NC  : {0}'.format(noNCframes_l)
            print 'Hilo ejecutandose?      : {0}'.format(hilorun_l)
            print 'Entro a esperar?        : {0}'.format(tiempo_l)

#Hilo para tramas sin NC
envio_sinNC = threading.Thread(target=sinNC)
envio_sinNC.start()

#Hilo para visualizacion
visualizador = threading.Thread(target=vista)
visualizador.start()

while True:

    #Recibir paquetes
    packet = s.recvfrom(65565)

    # Convertir la trama a string de bytes en Hexadecimal y obtener campos Ethernet
    packet = packet[0]
    trama_recibida = packet.encode('hex')
    destino = trama_recibida[0:12]
    origen = trama_recibida[12:24]

    #Solo se procesaran tramas que no provengan de manera local o del enlace con el Master
    if origen not in [MAC_SWITCH5, MAC_SWITCH5_U]:

        #Trama enviada desde el SW1, sentido de Tx ->
        if origen == MAC_HOST1:
            destino = MAC_HOST3
            frame_nchost1 += 1
            nchost1_frames[frame_nchost1] = destino + trama_recibida[12:]

        # Trama enviada desde el SW2, sentido de Tx ->
        if origen == MAC_HOST2:
            destino = MAC_HOST4
            frame_nchost2 += 1
            nchost2_frames[frame_nchost2] = destino + trama_recibida[12:]

        # Trama enviada desde el SW1, sentido de Tx <-
        if origen == MAC_HOST3:
            destino = MAC_HOST1
            frame_nchost3 += 1
            nchost3_frames[frame_nchost3] = destino + trama_recibida[12:]
            s.send(nchost3_frames[frame_nchost3].decode('hex'))

        # Trama enviada desde el SW2, sentido de Tx <-
        if origen == MAC_HOST4:
            destino = MAC_HOST2
            frame_nchost4 += 1
            nchost4_frames[frame_nchost4] = destino + trama_recibida[12:]
            s.send(nchost4_frames[frame_nchost4].decode('hex'))


        # Trama unica enviada desde el SW1, sentido de Tx <-
        if origen == MAC_HOST3_U:
            origen = MAC_HOST3
            destino = MAC_HOST1
            frame_nchost3 += 1
            frame_nchost4 += 1
            frame_ncswitch6 += 1
            nchost3_frames[frame_nchost3] = destino + origen + trama_recibida[24:]
            s.send(nchost3_frames[frame_nchost3].decode('hex'))

        # Trama unica enviada desde el SW2, sentido de Tx <-
        if origen == MAC_HOST4_U:
            origen = MAC_HOST4
            destino = MAC_HOST2
            frame_nchost3 += 1
            frame_nchost4 += 1
            frame_ncswitch6 += 1
            nchost4_frames[frame_nchost4] = destino + origen + trama_recibida[24:]
            s.send(nchost4_frames[frame_nchost4].decode('hex'))
```

```
        #Verificar si se pueden crear tramas codificadas
        if (frame_nchost1 > 0 or frame_nchost2 > 0) and origen in [MAC_HOST1, MAC_HOST2]:

            if frame_nchost1 == frame_nchost2 and frame_nchost1 > frame_ncswitch5 and
frame_nchost2 > frame_ncswitch5:

                frame_ncswitch5 += 1
                trama_codificada = '0000000000' +
str(format(int(nchost1_frames[frame_nchost1], 16) ^

int(nchost2_frames[frame_nchost2], 16), '02x'))
                ncswitch5_frames[frame_ncswitch5] = trama_codificada[0:12] + MAC_H1XORH2_88
+ trama_codificada[24:]

                #Enviar Tramas
                s.send(nchost1_frames[frame_nchost1].decode('hex'))
                s.send(nchost2_frames[frame_nchost2].decode('hex'))
                s.send(ncswitch5_frames[frame_ncswitch5].decode('hex'))

                espera = False
                reloj = clock()
                tiempo = 0

            else:

                if frame_nchost1 > frame_nchost2 == frame_ncswitch5:
                    espera = True

                if frame_nchost2 > frame_nchost1 == frame_ncswitch5:
                    espera = True

                if frame_nchost1 > frame_nchost2 > frame_ncswitch5:

                    espera = False
                    reloj = clock()
                    tiempo = 0

                    frame_ncswitch5 += 1

                    trama_codificada = '0000000000' +
str(format(int(nchost1_frames[frame_nchost2], 16) ^

int(nchost2_frames[frame_nchost2], 16), '02x'))
                    ncswitch5_frames[frame_ncswitch5] = trama_codificada[0:12] +
MAC_H1XORH2_88 + trama_codificada[24:]

                    # Enviar Tramas
                    s.send(nchost1_frames[frame_nchost2].decode('hex'))
                    s.send(nchost2_frames[frame_nchost2].decode('hex'))
                    s.send(ncswitch5_frames[frame_ncswitch5].decode('hex'))

                    espera = False
                    reloj = clock()
                    tiempo = 0

                if frame_nchost2 > frame_nchost1 > frame_ncswitch5:

                    espera = False
                    reloj = clock()
                    tiempo = 0

                    frame_ncswitch5 += 1

                    trama_codificada = '0000000000' +
str(format(int(nchost1_frames[frame_nchost1], 16) ^

int(nchost2_frames[frame_nchost1], 16), '02x'))
                    ncswitch5_frames[frame_ncswitch5] = trama_codificada[0:12] +
MAC_H1XORH2_88 + trama_codificada[24:]

                    # Enviar Tramas
                    s.send(nchost1_frames[frame_nchost1].decode('hex'))
                    s.send(nchost2_frames[frame_nchost1].decode('hex'))
```

```
                    s.send(ncswitch5_frames[frame_ncswitch5].decode('hex'))

                    espera = False
                    reloj = clock()
                    tiempo = 0

        #Tratamiento de trama NC
        if origen == MAC_H3XORH4_77:

            destino = MAC_H1XORH2_88
            frame_ncswitch6 += 1
            ncswitch6_frames[frame_ncswitch6] = destino + trama_recibida[12:]

            #Si se han recibido previamente tramas del Host 3 y 4
            if frame_nchost3 >= frame_ncswitch6 and frame_nchost4 >= frame_ncswitch6:

                espera = False
                reloj = clock()
                tiempo = 0

            else:

                espera = False
                reloj = clock()
                tiempo = 0

                recuperacion += 1

                #Se requiere recuperar la trama del Host 3

                if frame_nchost4 >= frame_ncswitch6 > frame_nchost3:

                    espera = False
                    reloj = clock()
                    tiempo = 0

                    trama_recuperada = '0000000000' +
str(format(int(ncswitch6_frames[frame_ncswitch6], 16) ^

int(nchost4_frames[frame_ncswitch6], 16), '02x'))

                    frame_h3 += 1
                    frame_nchost3 += 1
                    nchost3_frames[frame_ncswitch6] = MAC_HOST1+ trama_recuperada[12:]

                    # Enviar Tramas
                    s.send(nchost3_frames[frame_ncswitch6].decode('hex'))

                    frame_recuperados_total += 1
                    frames_recuperados_nchost3 += 1

                    espera = False
                    reloj = clock()
                    tiempo = 0

                # Se requiere recuperar la trama del Host 4

                if frame_nchost3 >= frame_ncswitch6 > frame_nchost4:

                    trama_recuperada = '0000000000' +
str(format(int(ncswitch6_frames[frame_ncswitch6], 16) ^

int(nchost3_frames[frame_ncswitch6], 16), '02x'))

                    frame_nchost4 += 1
                    nchost4_frames[frame_ncswitch6] = MAC_HOST2 + trama_recuperada[12:]

                    # Enviar Tramas
                    s.send(nchost4_frames[frame_ncswitch6].decode('hex'))

                    frame_recuperados_total += 1
                    frames_recuperados_nchost4 +=1

                    espera = False
```

```
        reloj = clock()
        tiempo = 0

    #No se puede recuperar tramas

    if frame_nchost3 < frame_ncswitch6 and frame_nchost4 < frame_ncswitch6:

        espera = False
        reloj = clock()
        tiempo = 0

        frame_ncswitch6 -= 1
        frame_perdidos +=1
```

ANEXO VI:

CÓDIGO DE LA APLICACIÓN PARA CODIFICAIÓN DE RED EN NC SWITCH6

```
import socket
import os
import threading
from time import clock
import ConfigParser

cfg = ConfigParser.ConfigParser()
archivo ="MACaddress.ini"
cfg.read(archivo)

#Importar direcciones MAC del archivo de configuracion
MAC_HOST1 = cfg.get('macs','mac_host1')
MAC_HOST2 = cfg.get('macs','mac_host2')
MAC_HOST3 = cfg.get('macs','mac_host3')
MAC_HOST4 = cfg.get('macs','mac_host4')
MAC_SWITCH5 = cfg.get('macs','mac_switch5')
MAC_SWITCH6 = cfg.get('macs','mac_switch6')
MAC_HOST1_U = cfg.get('macs','mac_host1_u')
MAC_HOST2_U = cfg.get('macs','mac_host2_u')
MAC_HOST3_U = cfg.get('macs','mac_host3_u')
MAC_HOST4_U = cfg.get('macs','mac_host4_u')
MAC_SWITCH5_U = cfg.get('macs','mac_switch5_u')
MAC_SWITCH6_U = cfg.get('macs','mac_switch6_u')
MAC_H3XORH4_77 = cfg.get('macs','mac_h3xorh4_77')
MAC_H1XORH2_88 = cfg.get('macs','mac_h1xorh2_88')

# Creacion del socket para capturar tramas
s = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0003))
s.bind(('ens33', 0))

print '\nSocket Creado Correctamente...'
print '\nEscuchando por Paquetes...'

# Tabla de Paquetes para cada Switch
nchost1_frames  = {}
nchost2_frames  = {}
nchost3_frames  = {}
nchost4_frames  = {}

# Tabla de Paquetes de NetworkCoding
ncswitch5_frames  = {}
ncswitch6_frames  = {}

#ID de cada trama de cada switch
frame_nchost1 = 0
frame_nchost2 = 0
frame_nchost3 = 0
frame_nchost4 = 0
frame_ncswitch5 = 0
frame_ncswitch6 = 0

#Contadores para estadisticas:
frame_recuperados_total = 0
frame_recuperados_nchost1 = 0
frame_recuperados_nchost2 = 0
frame_perdidos = 0

recuperacion = 0

#Variable para determinar si es necesario enviar paquetes sobrantes
espera = False
noNCframes = 0
hilorun = 0
tiempo = 0
```

```
sw1_noNC = 0
sw2_noNC = 0
sw4_noNC = 0
sw3_noNC = 0

#Variable para reloj
reloj = clock()

#Temporizados para enviar paquetes sobrantes
def temporizador():

    global hilorun
    global tiempo
    global noNCframes
    global sw1_noNC
    global sw2_noNC
    global sw3_noNC
    global sw4_noNC
    global frame_nchost1
    global frame_nchost2
    global frame_nchost3
    global frame_nchost4
    global frame_ncswitch5
    global frame_ncswitch6
    global reloj
    global espera

    hilorun = 1
    x = 1

    while True:

        reloj = clock()

        while espera:
            tiempo = 1
            if clock() - reloj > 1 and espera:
                if frame_nchost3 > frame_nchost4 == frame_ncswitch6 and espera and clock()
- reloj > 1:

                    trama_sw3 = nchost3_frames [frame_nchost3]
                    trama_sw3 = trama_sw3[0:12] + MAC_HOST3_U + trama_sw3[24:]
                    s.send(trama_sw3.decode('hex'))

                    frame_nchost4 += 1
                    frame_ncswitch6 += 1
                    noNCframes += 1
                    sw3_noNC += 1
                    if frame_nchost3 == frame_nchost4:
                        espera = False
                        tiempo = 0
                if frame_nchost4 > frame_nchost3 == frame_ncswitch6 and espera and clock()
- reloj > x:

                    trama_sw4 = nchost4_frames [frame_nchost4]
                    trama_sw4 = trama_sw4[0:12] + MAC_HOST4_U + trama_sw4[24:]
                    s.send(trama_sw4.decode('hex'))

                    frame_nchost3 += 1
                    frame_ncswitch6 += 1
                    noNCframes += 1
                    sw4_noNC += 1
                    if frame_nchost4 == frame_nchost3:
                        espera = False
                        tiempo = 0
                reloj = clock()


def vista():

    global frame_recuperados_nchost1
    global frame_recuperados_nchost2
    global frame_perdidos
    global recuperacion
```

```
global noNCframes
global hilorun
global tiempo
global frame_nchost1
global frame_nchost2
global frame_nchost3
global frame_nchost4
global frame_ncswitch5
global frame_ncswitch6

frame_recuperados_nchost1_l = frame_recuperados_nchost1
frame_recuperados_nchost2_l = frame_recuperados_nchost2
frame_perdidos_l = frame_perdidos
recuperacion_l = recuperacion
noNCframes_l = noNCframes
hilorun_l = hilorun
tiempo_l = tiempo
frame_nchost1_l = frame_nchost1
frame_nchost2_l = frame_nchost2
frame_nchost3_l = frame_nchost3
frame_nchost4_l = frame_nchost4
frame_ncswitch5_l = frame_ncswitch5
frame_ncswitch6_l = frame_ncswitch6

os.system('clear')
print '\nEstadisticas de Paquetes: \n'
print 'Paquetes RecuperadosSW1 : {0}'.format(frame_recuperados_nchost1_l)
print 'Paquetes RecuperadosSW2 : {0}'.format(frame_recuperados_nchost2_l)
print 'Paquetes Perdidos       : {0}'.format(frame_perdidos_l)
print 'Recuperacion            : {0}'.format(recuperacion_l)
print 'Tramas enviadas sin NC  : {0}'.format(noNCframes_l)
print 'Hilo ejecutandose?      : {0}'.format(hilorun_l)
print 'Entro a esperar?        : {0}'.format(tiempo_l)
print '\nVariables frame sw: \n'
print 'frame_nchost1: {0}'.format(frame_nchost1_l)
print 'frame_nchost2: {0}'.format(frame_nchost2_l)
print 'frame_nchost3: {0}'.format(frame_nchost3_l)
print 'frame_nchost4: {0}'.format(frame_nchost4_l)
print 'frame_ncswitch5: {0}'.format(frame_ncswitch5_l)
print 'frame_ncswitch6: {0}'.format(frame_ncswitch6_l)

while True:

    if frame_recuperados_nchost1_l != frame_recuperados_nchost1 or \
                    frame_recuperados_nchost2_l != frame_recuperados_nchost2 or \
                    frame_perdidos_l != frame_perdidos or \
                    recuperacion_l != recuperacion or \
                    noNCframes_l != noNCframes or \
                    hilorun_l != hilorun or \
                    tiempo_l != tiempo or \
                    frame_nchost1_l != frame_nchost1 or \
                    frame_nchost2_l != frame_nchost2 or \
                    frame_nchost3_l != frame_nchost3 or \
                    frame_nchost4_l != frame_nchost4 or \
                    frame_ncswitch5_l != frame_ncswitch5 or \
                    frame_ncswitch6_l != frame_ncswitch6:

        frame_recuperados_nchost1_l = frame_recuperados_nchost1
        frame_recuperados_nchost2_l = frame_recuperados_nchost2
        frame_perdidos_l = frame_perdidos
        recuperacion_l = recuperacion
        noNCframes_l = noNCframes
        hilorun_l = hilorun
        tiempo_l = tiempo
        frame_nchost1_l = frame_nchost1
        frame_nchost2_l = frame_nchost2
        frame_nchost3_l = frame_nchost3
        frame_nchost4_l = frame_nchost4
        frame_ncswitch5_l = frame_ncswitch5
        frame_ncswitch6_l = frame_ncswitch6

        os.system('clear')
        print '\nEstadisticas de Paquetes: \n'
        print 'Paquetes RecuperadosSW1 : {0}'.format(frame_recuperados_nchost1_l)
```

```
            print 'Paquetes RecuperadosSW2 : {0}'.format(frame_recuperados_nchost2_l)
            print 'Paquetes Perdidos       : {0}'.format(frame_perdidos_l)
            print 'Recuperacion            : {0}'.format(recuperacion_l)
            print 'Tramas enviadas sin NC  : {0}'.format(noNCframes_l)
            print 'Hilo ejecutandose?      : {0}'.format(hilorun_l)
            print 'Entro a esperar?        : {0}'.format(tiempo_l)
            print '\nVariables frame sw: \n'
            print 'frame_nchost1: {0}'.format(frame_nchost1_l)
            print 'frame_nchost2: {0}'.format(frame_nchost2_l)
            print 'frame_nchost3: {0}'.format(frame_nchost3_l)
            print 'frame_nchost4: {0}'.format(frame_nchost4_l)
            print 'frame_ncswitch5: {0}'.format(frame_ncswitch5_l)
            print 'frame_ncswitch6: {0}'.format(frame_ncswitch6_l)


# Hilo para paquetes sobrantes
hilo = threading.Thread(target=temporizador)
hilo.start()

# Hilo para visualizacion
ver = threading.Thread(target=vista)
ver.start()

while True:

    #Recibir paquetes
    packet = s.recvfrom(65565)

    # Convertir la trama a string de bytes en Hexadecimal y obtener campos Ethernet
    packet = packet[0]
    trama_recibida = packet.encode('hex')
    destino = trama_recibida[0:12]
    origen = trama_recibida[12:24]

    #Solo se procesaran tramas que no provengan de manera local o del enlace con el Master
    if origen not in [MAC_SWITCH6, MAC_SWITCH6_U]:

        #Trama enviada desde el SW3, sentido de Tx ->
        if origen == MAC_HOST1:
            destino = MAC_HOST3
            frame_nchost1 += 1
            nchost1_frames [frame_nchost1] = destino + trama_recibida[12:]
            s.send(nchost1_frames [frame_nchost1].decode('hex'))

        # Trama enviada desde el SW4, sentido de Tx ->
        if origen == MAC_HOST2:
            destino = MAC_HOST4
            frame_nchost2 += 1
            nchost2_frames [frame_nchost2] = destino + trama_recibida[12:]
            s.send(nchost2_frames [frame_nchost2].decode('hex'))

        # Trama enviada desde el SW3, sentido de Tx <-
        if origen == MAC_HOST3:
            destino = MAC_HOST1
            frame_nchost3 += 1
            nchost3_frames [frame_nchost3] = destino + trama_recibida[12:]

        # Trama enviada desde el SW4, sentido de Tx <-
        if origen == MAC_HOST4:
            destino = MAC_HOST2
            frame_nchost4 += 1
            nchost4_frames [frame_nchost4] = destino + trama_recibida[12:]

        # Trama unica enviada desde el SW3, sentido de Tx ->
        if origen == MAC_HOST1_U:
            origen = MAC_HOST1
            destino = MAC_HOST3
            frame_nchost1 += 1
            frame_nchost2 += 1
            frame_ncswitch5 += 1
            nchost1_frames [frame_nchost1] = destino + origen + trama_recibida[24:]
            s.send(nchost1_frames [frame_nchost1].decode('hex'))

        # Trama unica enviada desde el SW4, sentido de Tx ->
```

```
        if origen == MAC_HOST2_U:
            origen = MAC_HOST2
            destino = MAC_HOST4
            frame_nchost1 += 1
            frame_nchost2 += 1
            frame_ncswitch5 += 1
            nchost2_frames [frame_nchost2] = destino + origen + trama_recibida[24:]
            s.send(nchost2_frames [frame_nchost2].decode('hex'))

        #Verificar si se pueden crear tramas codificadas
        if (frame_nchost3 > 0 or frame_nchost4 > 0) and origen in [MAC_HOST3, MAC_HOST4]:

            if frame_nchost3 == frame_nchost4 and frame_nchost3 > frame_ncswitch6 and
frame_nchost4 > frame_ncswitch6:

                frame_ncswitch6 += 1
                trama_codificada = '0000000000' + str(format(int(nchost3_frames
[frame_nchost3], 16) ^
                                                             int(nchost4_frames
[frame_nchost4], 16), '02x'))
                ncswitch6_frames [frame_ncswitch6] = MAC_H1XORH2_88 + trama_codificada[12:]

                #Enviar Tramas
                s.send(nchost3_frames [frame_nchost3].decode('hex'))
                s.send(nchost4_frames [frame_nchost4].decode('hex'))
                s.send(ncswitch6_frames [frame_ncswitch6].decode('hex'))

                espera = False
                reloj = clock()
                tiempo = 0
            else:

                if frame_nchost3 > frame_nchost4 == frame_ncswitch6:
                    espera = True

                if frame_nchost4 > frame_nchost3 == frame_ncswitch6:
                    espera = True

                if frame_nchost3 > frame_nchost4 > frame_ncswitch6:

                    espera = False
                    reloj = clock()
                    tiempo = 0

                    frame_ncswitch6 += 1

                    trama_codificada = '0000000000' + str(format(int(nchost3_frames
[frame_nchost4], 16) ^
                                                                 int(nchost4_frames
[frame_nchost4], 16), '02x'))
                    ncswitch6_frames [frame_ncswitch6] = MAC_H1XORH2_88  +
trama_codificada[12:]

                    # Enviar Tramas
                    s.send(nchost3_frames [frame_nchost4].decode('hex'))
                    s.send(nchost4_frames [frame_nchost4].decode('hex'))
                    s.send(ncswitch6_frames [frame_ncswitch6].decode('hex'))

                    espera = False
                    reloj = clock()
                    tiempo = 0

                if frame_nchost4 > frame_nchost3 > frame_ncswitch6:

                    espera = False
                    reloj = clock()
                    tiempo = 0

                    frame_ncswitch6 += 1

                    trama_codificada = '0000000000' + str(format(int(nchost3_frames
[frame_nchost3], 16) ^
                                                                 int(nchost4_frames
[frame_nchost3], 16), '02x'))
```

```
                    ncswitch6_frames [frame_ncswitch6] = MAC_H1XORH2_88 +
trama_codificada[24:]

                    # Enviar Tramas
                    s.send(nchost3_frames [frame_nchost3].decode('hex'))
                    s.send(nchost4_frames [frame_nchost3].decode('hex'))
                    s.send(ncswitch6_frames [frame_ncswitch6].decode('hex'))

                    espera = False
                    reloj = clock()
                    tiempo = 0

        #Tratamiento de trama NC
        if origen == MAC_H1XORH2_88:

            espera = False
            reloj = clock()
            tiempo = 0

            destino = MAC_H3XORH4_77
            frame_ncswitch5 += 1
            ncswitch5_frames [frame_ncswitch5] = destino + trama_recibida[12:]

            #Si se han recibido previamente tramas del Host 1 y 2

            if frame_nchost1 >= frame_ncswitch5 and frame_nchost2 >= frame_ncswitch5:

                espera = False
                reloj = clock()
                tiempo = 0

            else:

                espera = False
                reloj = clock()
                tiempo = 0

                recuperacion += 1

                #Se requiere recuperar la trama del Host 1
                if frame_nchost2 >= frame_ncswitch5 > frame_nchost1:

                    espera = False
                    reloj = clock()
                    tiempo = 0

                    trama_recuperada = '0000000000' + str(format(int(ncswitch5_frames
[frame_ncswitch5], 16) ^
                                                                 int(nchost2_frames
[frame_ncswitch5], 16), '02x'))

                    frame_nchost1 += 1
                    nchost1_frames [frame_ncswitch5] = trama_recuperada[0:12] + MAC_HOST1 +
trama_recuperada[24:]

                    # Enviar Tramas
                    s.send(nchost1_frames [frame_ncswitch5].decode('hex'))

                    frame_recuperados_total += 1
                    frame_recuperados_nchost1 += 1

                    espera = False
                    reloj = clock()
                    tiempo = 0

                # Se requiere recuperar la trama del Host 2

                if frame_nchost1 >= frame_ncswitch5 > frame_nchost2:

                    trama_recuperada = '0000000000' + str(format(int(ncswitch5_frames
[frame_ncswitch5], 16) ^
                                                                 int(nchost1_frames
[frame_ncswitch5], 16), '02x'))
```

```
                    frame_nchost2 += 1
                    nchost2_frames [frame_ncswitch5] = trama_recuperada[0:12] + MAC_HOST2 +
trama_recuperada[24:]

                    # Enviar Tramas
                    s.send(nchost2_frames [frame_ncswitch5].decode('hex'))

                    frame_recuperados_total += 1
                    frame_recuperados_nchost2 +=1

                    espera = False
                    reloj = clock()
                    tiempo = 0

                #No se puede recuperar tramas

                if frame_nchost1 < frame_ncswitch5 and frame_nchost2 < frame_ncswitch5:

                    espera = False
                    reloj = clock()
                    tiempo = 0

                    frame_ncswitch5 -= 1
                    frame_perdidos +=1
```

ANEXO VII:
MANUAL DE USUARIO

Se debe aclarar que se debe disponer de 7 máquinas virtuales de acuerdo a las especificaciones presentadas en el Capítulo 2 del presente trabajo.

En la máquina virtual **NC Master** se debe tener instalado Mininet y el controlador Ryu; y debe tener los archivos: `RedTopoloiaButterfly.py`, `ReglasTransmisionNormal.py`, `ReglasCodificacionRed.py`, `Pérdidas.ini`, `MACaddress.ini`, `RYUMACaddress.ini`.

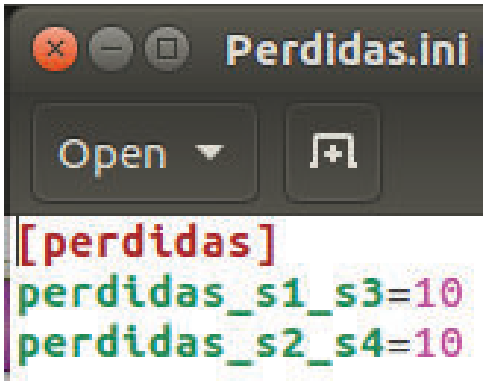
Cada uno de los equipos terminales **NC Host1**, **NC Host2**, **NC Host3**, y **NC Host4** deben poseer conectividad con **NC Master** y tener instalado Wireshark.

Además, **NC Host3** y **NC Host4** deben tener los scripts `ICMP_Host3.sh` y `ICMP_Host4.sh` respectivamente para facilitar las pruebas.

NC Switch5 y **NC Switch6** deben tener los archivos `CodificadorRedSwitch5.py` y `CodificadorRedSwitch6.py` respectivamente, así como también tener instalado Wireshark.

Una vez iniciadas las 7 máquinas virtuales se procede de la siguiente manera:

1. Abrir el archivo `Perdidas.ini` en **NC Master** y especificar los parámetros de porcentajes de pérdidas para los enlaces s1-s3 y s2-s4. Se debe especificar con números enteros entre 0 y 100 como se muestra a continuación:



2. Guardar los cambios en el archivo `Perdidas.ini` y cerrar.

3. En cada uno de los equipos terminales **NC Host1**, **NC Host2**, **NC Host3**, y **NC Host4** crear de forma manual entradas en las tablas ARP que contengan las direcciones IP especificadas en el Capítulo 2 en cada uno de los equipos terminales utilizando el comando:

```
Sudo arp -i [interfaz de red] -s [dirección ip] [dirección MAC]
```

Se debe verificar que se haya ingresado correctamente la información utilizando el comando `sudo arp`. Para cada terminal, el ingreso de la asociación ARP y la verificación del resultado de la operación se muestra a continuación:

```
nchost1@ubuntu:~$ sudo arp -i ens33 -s 192.168.30.33 00:00:00:00:00:33
[sudo] password for nchost1:
nchost1@ubuntu:~$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.168.2            ether   00:50:56:f8:99:d4   C                     ens34
192.168.168.254          ether   00:50:56:e5:ae:56   C                     ens34
192.168.30.33            ether   00:00:00:00:00:33   CM                    ens33
nchost2@ubuntu:~$ sudo arp -i ens33 -s 192.168.40.44 00:00:00:00:00:44
[sudo] password for nchost2:
nchost2@ubuntu:~$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.168.2            ether   00:50:56:f8:99:d4   C                     ens34
192.168.40.44            ether   00:00:00:00:00:44   CM                    ens33
192.168.168.254          ether   00:50:56:e5:ae:56   C                     ens34
nchost3@ubuntu:~$ sudo arp -i ens33 -s 192.168.10.11 00:00:00:00:00:11
[sudo] password for nchost3:
nchost3@ubuntu:~$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.168.2            ether   00:50:56:f8:99:d4   C                     ens34
192.168.10.11            ether   00:00:00:00:00:11   CM                    ens33
192.168.168.254          ether   00:50:56:e5:ae:56   C                     ens34
nchost4@ubuntu:~$ sudo arp -i ens33 -s 192.168.20.22 00:00:00:00:00:22
[sudo] password for nchost4:
nchost4@ubuntu:~$ sudo arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.168.2            ether   00:50:56:f8:99:d4   C                     ens34
192.168.168.254          ether   00:50:56:e5:ae:56   C                     ens34
192.168.20.22            ether   00:00:00:00:00:22   CM                    ens33
```

4. Sincronizar los relojes del sistema de cada terminal usando NTP con el comando `ntp [url del servidor]`, utilizando un servidor público para cada terminal el resultado de esta operación genera el siguiente resultado:

```
nchost1@ubuntu:~$ sudo ntpdate -u ntp.ubuntu.com
30 Jan 10:53:21 ntpdate[3349]: adjust time server 91.189.91.157 offset -0.007270 sec
nchost2@ubuntu:~$ sudo ntpdate -u ntp.ubuntu.com
30 Jan 10:53:32 ntpdate[3337]: adjust time server 91.189.91.157 offset 0.002196 sec
```

```
nchost3@ubuntu:~$ sudo ntpdate -u ntp.ubuntu.com
30 Jan 10:53:43 ntpdate[3312]: adjust time server 91.189.91.157 offset -0.020428 sec
nchost4@ubuntu:~$ sudo ntpdate -u ntp.ubuntu.com
30 Jan 10:53:54 ntpdate[3315]: adjust time server 91.189.91.157 offset -0.002240 sec
```

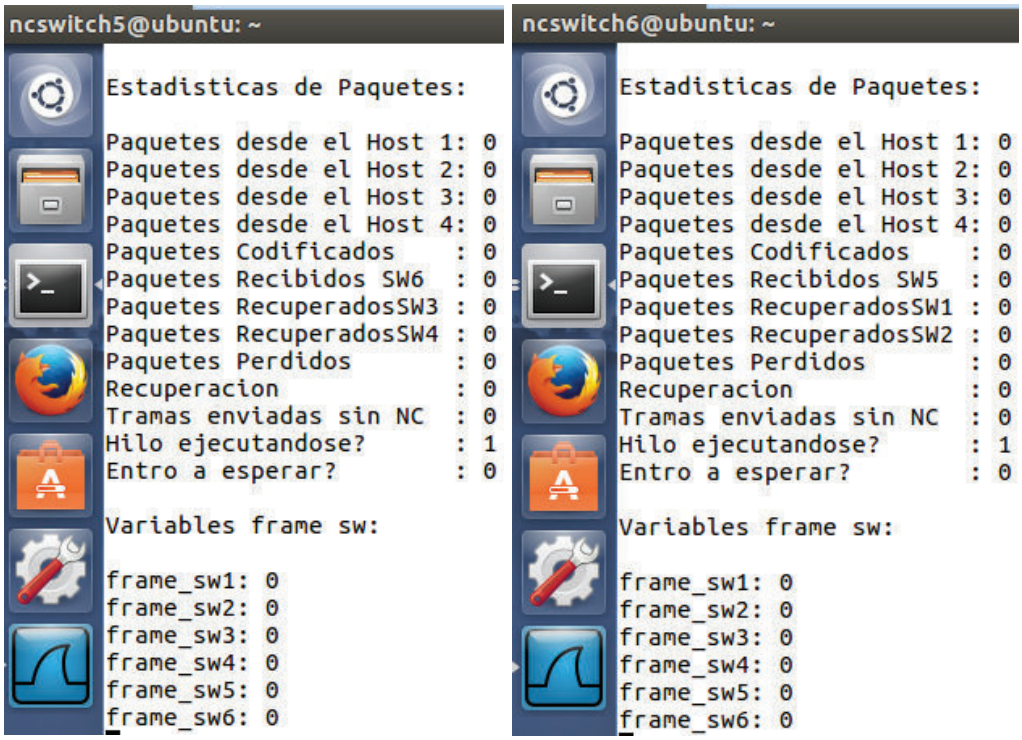
En un terminal de Linux de **NC Master** ejecutar el archivo `RedTopologíaButterfly.py` con el comando `sudo python [path completo del archivo].` El resultado de esta operación se muestra a continuación:

```
ncmaster@ubuntu:~$ sudo python ./Dropbox/Tesis/NC\ Tesis/TopologiaIdeal.py
[sudo] password for ncmaster:
Connecting to remote controller at 127.0.0.1:6653
*** Creating network
*** Adding controller
*** Adding hosts:

*** Adding switches:
s1 s2 s3 s4 s5 s6
*** Adding links:
(100.00Mbit 20ms delay) (100.00Mbit 20ms delay) (s1, s3) (100.00Mbit 1ms de
lay) (100.00Mbit 1ms delay) (s1, s5) (100.00Mbit 20ms delay) (100.00Mbit 20
ms delay) (s2, s4) (100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (s2, s5) (
100.00Mbit 1ms delay) (100.00Mbit 1ms delay) (s3, s6) (100.00Mbit 1ms delay
) (100.00Mbit 1ms delay) (s4, s6) (100.00Mbit 30ms delay) (100.00Mbit 30ms
delay) (s5, s6)
*** Configuring hosts

*** Adding hardware interface ens33 to switch s1
*** Adding hardware interface ens34 to switch s2
*** Adding hardware interface ens35 to switch s3
*** Adding hardware interface ens36 to switch s4
*** Adding hardware interface ens36 to switch s5
*** Adding hardware interface ens36 to switch s6
*** Starting controller
c0
*** Starting 6 switches
s1 s2 s3 s4 s5 s6 ...(100.00Mbit 20ms delay) (100.00Mbit 1ms delay) (100.00
Mbit 20ms delay) (100.00Mbit 1ms delay) (100.00Mbit 20ms delay) (100.00Mbit
 1ms delay) (100.00Mbit 20ms delay) (100.00Mbit 1ms delay) (100.00Mbit 1ms
delay) (100.00Mbit 1ms delay) (100.00Mbit 30ms delay) (100.00Mbit 1ms delay
) (100.00Mbit 1ms delay) (100.00Mbit 30ms delay)
*** Starting CLI:
mininet>
```

5. Ejecutar la aplicación para el controlador, ya sea `ReglasTrasmisionNormal.py` o `ReglasCodificacionRed.py`, en otra terminal de **NC Master** usando el intérprete de Python utilizando el comando `PYTHONPATH=. [path completo del archivo]` lo cual genera el siguiente resultado que indica una correcta conexión entre el controlador y el ambiente de red vitrtual:

```
ncmaster@ubuntu:~/ryu$ PYTHONPATH=. ./bin/ryu-manager ~/Dropbox/Tesis/NC\ Tesis/Netw
orkCodingController.py
loading app /home/ncmaster/Dropbox/Tesis/NC Tesis/NetworkCodingController.py
loading app ryu.controller.ofp_handler
instantiating app /home/ncmaster/Dropbox/Tesis/NC Tesis/NetworkCodingController.py o
f SwitchNC
instantiating app ryu.controller.ofp_handler of OFPHandler
--------------RED CONFIGURADA CORRECTAMENTE-------------
```

6. Ejecutar las aplicaciones `CodificadorRedSwitch5.py` y `CodificadorSwitch6.py` en **NC Switch5** y **NC Switch6** usando el intérprete de Python con el comando `python path completo del archivo]`, obteniendo como resultado las pantallas siguientes:



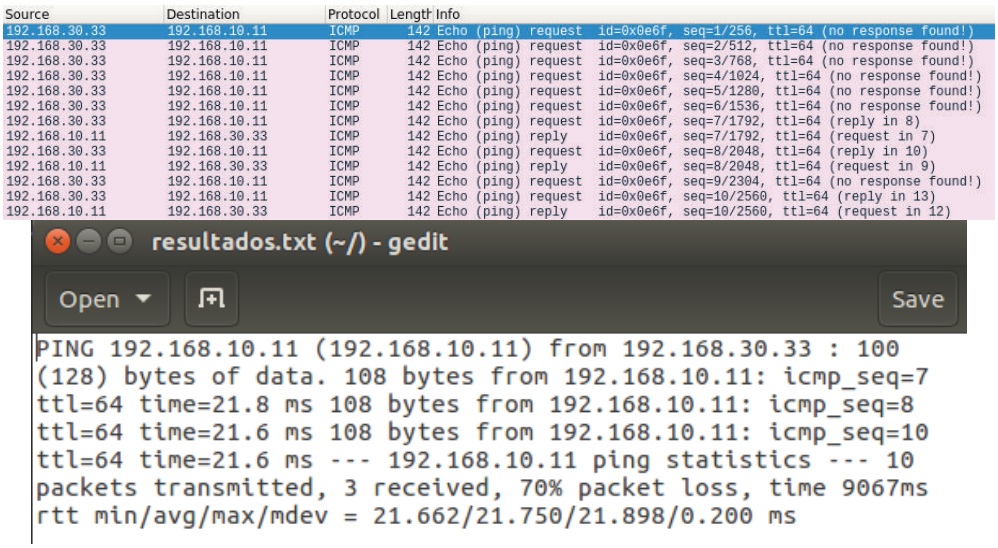
7. Editar los *scripts* de pruebas (`ICMP_Host3.sh` y `ICMP_Host4.sh`) en **NC Host3** y **NC Host4** y especificar el número y el tamaño de paquetes para la prueba. El tamaño del paquete se especifica luego del parámetro `-s` y la cantidad de paquetes luego del parámetro `-c` como se muestra a continuación:

```
#! /bin/bash
echo $(ping -s 100 -I 192.168.30.33 -c 10 192.168.10.11) > resultados.txt
```

```
#! /bin/bash
echo $(ping -s 100 -I 192.168.40.44 -c 10 192.168.20.22) > resultados.txt
```

8. Ejecutar los *scripts* utilizando el comando `at` en **NC Host3** y **NC Host4** usando la sintaxis: `at [hora] -f [script]`. Recordar poner la misma hora de ejecución en ambos terminales.

9. Abrir Wireshark en cada terminal y esperar que la prueba se ejecute. Wireshark capturará todas las tramas que circulen en cada terminal y los archivos `resultados.txt` mostrarán los resultados de la ejecución del ping para **NC Host3** y **NC Host4.**



Source	Destination	Protocol	Length	Info
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=1/256, ttl=64 (no response found!)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=2/512, ttl=64 (no response found!)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=3/768, ttl=64 (no response found!)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=4/1024, ttl=64 (no response found!)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=5/1280, ttl=64 (no response found!)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=6/1536, ttl=64 (no response found!)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=7/1792, ttl=64 (reply in 8)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0e6f, seq=7/1792, ttl=64 (request in 7)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=8/2048, ttl=64 (reply in 10)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0e6f, seq=8/2048, ttl=64 (request in 9)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=9/2304, ttl=64 (no response found!)
192.168.30.33	192.168.10.11	ICMP	142	Echo (ping) request id=0x0e6f, seq=10/2560, ttl=64 (reply in 13)
192.168.10.11	192.168.30.33	ICMP	142	Echo (ping) reply id=0x0e6f, seq=10/2560, ttl=64 (request in 12)