IES Palas Atenea

Proyecto de Investigación Bachillerato de excelencia:

Programación, redes y software libre

Invproy α.zip

Un simulador de redes por y para alumnos

Adaptación para el *II Encuentro Preuniversitario Jóvenes Investigadores*

David Davó Laviña

Tutor
Julio Sánchez Olías

17 de enero de 2017

Índice general

Índice de figuras

Introducción

En el mundo contemporáneo, ninguna de las innovaciones tecnológicas sería posible sin algo fundamental: las redes; y, más concretamente, redes informáticas. Las redes informáticas han hecho posible, desde su nacimiento, la comunicación de grandes sumas de datos a velocidades casi instantáneas entre sitios distantes. Al principio esta tecnología era usada entre universidades, acelerando el proceso de investigación al coordinarse unas universidades con otras mucho más rápidamente.

Más tarde, se extendió el uso de esta tecnología del uso militar y científico a todas las empresas y hogares, comenzando así una revolución tecnológica que aún no se ha conseguido parar. Con acceso instantáneo a cultura, entretenimiento, conocimiento, información y más de dos mil exabytes de ancho de banda[1] viajando por la red, se ha convertido en una herramienta de uso por la humanidad imprescindible para cualquier actividad.

La tecnología del mundo contemporáneo no habría sido posible, en parte, también gracias al software libre y al desarrollo colaborativo, pues ha permitido el desarrollo de sistemas operativos como GNU/Linux de la *Free Software Foundation* (Usado actualmente por el 90 % de los servidores de red) o CUPS, el software para servidores de administración de impresoras más completo y competente usado en la mayoría de oficinas.

Son dos cosas muy importantes, que apenas son enseñadas en las clases de la ESO y Bachillerato, por eso he creado InvProy, un pequeño simulador de redes con la ambición de enseñar tanto de redes como de programación. Podrán experimentar, de una forma sencilla y muy visual como funciona una red y cómo se comportan los distintos protocolos. También, al ser software libre, los alumnos podrán aprender sobre programación al observar el código y tener la licencia para modificarlo y colaborar en el desarrollo del programa. Aunque el programa este aún en fase Alpha (fase de desarrollo), ya tiene la base para que sea muy sencillo añadir más protocolos, funcionalidades o dispositivos de red. A día de hoy tiene como dispositivos los ordenadores, conmutadores y concentradores. En cuanto a paquetes de red, permite enviar un Ping, usando los protocolos ICMP, IPv4 y Ethernet.

[1] Datos a Mayo de 2015. Fuente: Cisco–[1]

1. Programación y software libre

Propuesta

El objetivo de este proyecto es el desarrollo abierto y colaborativo a largo plazo de un software programado en Python de código libre con el que los alumnos puedan aprender tanto sobre redes como de programación. Debe soportar los protocolos más utilizados en la actualidad y permitir una gran personalización por los usuarios. Además debe ser compatible con los sistemas operativos Ubuntu, MaX y Windows, y ser de fácil instalación para el alumnado. Debe ser intuitivo y fácil de usar e incluir una completa documentación.

1.1 Software Libre

Según la Free Software Foundation "«Software libre» es el software que respeta la libertad de los usuarios y la comunidad. A grandes rasgos, significa que los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software. Es decir, el «software libre» es una cuestión de libertad, no de precio. Para entender el concepto, piense en «libre» como en «libre expresión», no como en «barra libre». En inglés a veces decimos «libre software», en lugar de «free software», para mostrar que no queremos decir que es gratuito." – R. Stallman [2]

No debemos confundir 'Software Libre' con 'Código abierto', ya que, aunque el código pueda ser leído por todo el mundo no significa que el resto de personas tengan licencia para redistribuir y/o editar el código. Software libre es el que cumple: [3], [4]

- **Libertad 0:** La libertad de ejecutar el programa cuando quieras, para cualquier propósito.
- **Libertad 1:** La libertad de estudiar cómo el programa funciona, y la posibilidad de cambiarlo para que se ejecute como tú deseas. (Acceso al código del programa).
- **Libertad 2:** La libertad de redistribuir las copias para ayudar a tus colegas.
- **Libertad 3:** La libertad de distribuir copias de tu versión modificada a otras personas.

Una de las grandes ventajas del software libre, aparece en la educación. Es muy útil para aprender ya que, si un alumno tiene curiosidad sobre el programa que está usando, puede consultar el código fuente en internet. Además, al ser licencias gratuitas, se puede destinar ese presupuesto a otras áreas como el hardware o el profesorado. También es útil en el desarrollo, pues cualquier programador puede solucionar un error que afecta a todos los usuarios.

2. Redes Informáticas

2.1 Capas de Red/Modelo OSI

El modelo OSI es un modelo de referencia para redes basado en capas de abstracción. Su objetivo es conseguir la interoperabilidad entre sistemas haciendo uso de los protocolos estandarizados. Fue creado en 1980 por la Organización Internacional de Estandarización (ISO). No es considerado una arquitectura de red porque los protocolos no forman parte del modelo, sino que son entidades de distintas normativas internacionales.

Capa	PDU[1]	Función	Ejemplos
1. Física	Bit	Transmisión y recepción de bits físicos sobre un medio físico (topología de red)	RJ45, IEEE 802.11, etc.
2. Data Link	Frame	Transmisión segura de *frames* entre dos nodos conectados por una capa física.	Ethernet, 802.11, etc...
3. Red	Paquete	Estructurar y administrar una red multinodo. Incluye enrutamiento, control de tráfico, y asignación de direcciones	IPv4, IPv6, ICMP...
4. Transporte	Datagrama(UDP) Segmento(TCP)	Transmisión de segmentos de datos entre los puntos de una red, incluyendo ACK	TCP, UDP...
5. Sesión	Datos	Administración de sesiones de comunicación, como intercambio continúo de información entre dos nodos.	SSH, RPC, PAP...
6. Presentación	Datos	Translación de datos entre un servicio de red y una aplicación. Incluye comprensión, encriptación/decriptación, y codificación de carácteres.	MIME, TLS
7. Aplicación	Datos	APIs de alto nivel, incluyendo recursos compartidos y acceso remoto de archivos	HTTP, FTP, SMTP...

[1] *Protocol Data Unit* o Unidad de Datos de Protocolo.

2.2 Paquetes de red

Son cada serie de bits en los que se divide la información enviada por una red.
Según el modelo OSI, un paquete es estrictamente el PDU de la capa de red. El paquete de red se encuentra encapsulado en la capa anterior del modelo OSI. Por ejemplo, en éstandares de comunicación TCP/IP, un segmento TCP puede ser llevado por varios paquetes IP transportados por varios frames de Ethernet. Es parecido a las unidades lingüísticas.
Está formado por varios protocolos y en él se distinguen tres partes:

Header o cabecera: Datos e información sobre el paquete. (Dirección IP, MAC, etc)

Payload o carga: Los datos que se quieren transferir.

Trailer o cola: En ocasiones es inexistente (como en UDP) pero suele ser un código de comprobación de errores.

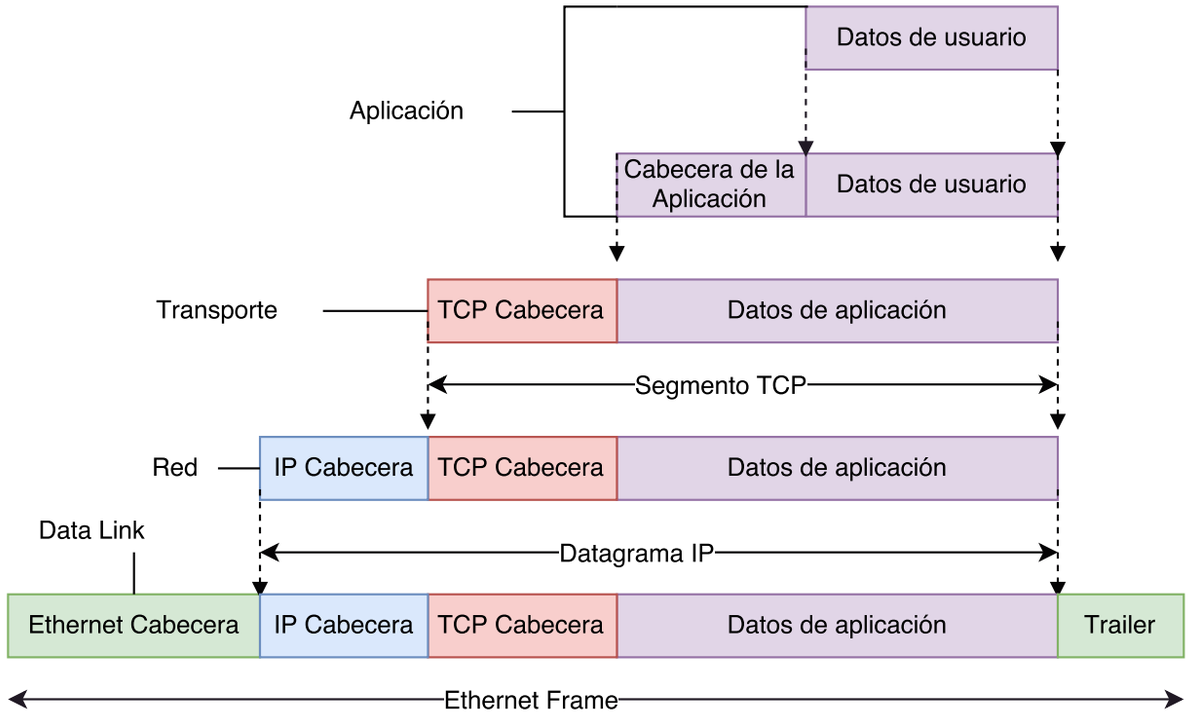

Figura 2.1: Encapsulación de red. El Datagrama IP es lo considerado 'Paquete de red'

2.3 Protocolos

Un protocolo de comunicación es un conjunto de reglas para intercambiar información entre enlaces de red. En una pila de protocolos, cada protocolo cubre los servicios del protocolo de la capa anterior. Por ejemplo, un e-mail se envía mediante el protocolo POP3 (*Post Office Protocol*, Protocolo de Oficina Postal) en la capa de Aplicación, sobre TCP en la capa de transporte, sobre IP en la capa de Red, sobre Ethernet para la capa *Data Link*, que está formado por bits. Para entenderlo mejor, es como la gramática de la lengua. Un sustantivo forma parte de un sintagma nominal, que forma parte de un sujeto, que a su vez forma parte de una oración. Siendo las ondas sonoras producidas la capa física y fonemas los bits.

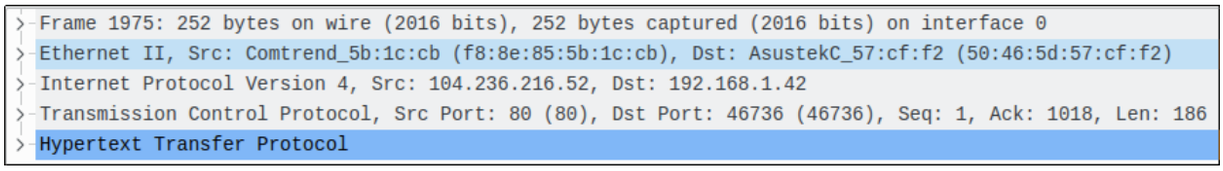

Figura 2.2: Captura de pantalla de Wireshark (Véase 3.1, pg. 8) en la que se muestran los protocolos que forman un paquete de red HTTP.

2.3.1 Familia de protocolos de internet

También conocido como *Internet Protocol Suite*, y más conocido como TCP/IP, es el fundamento de las redes informáticas. Se trata de un conjunto de más de 100 protocolos que permiten la interconexión de ordenadores, incluyendo protocolos de las aplicaciones más usadas.

2.3.1.1 Aplicación

Es la capa en la que se envían los datos a otras aplicaciones en otro ordenador o en el mismo. Las aplicaciones hacen uso de las capas inferiores para asegurarse que los datos lleguen a su destino. Algunos de los protocolos más usados son:

- **HTTP** *Hypertex Transfer Protocol:* Protocolo de Transferencia de Hipertexto. Es el protocolo base de la World Wide Web. Se trata de texto estructurado que usa hiperenlaces entre nodos que también contienen texto. El cliente, al entrar en una URL (*Uniform Resource Identifier*, Identificador de Recursos Uniforme) a través del agente de usuario (navegador) envía al servidor una petición. El servidor envía como respuesta un documento HTML u otro recurso.
- **DNS** *Domain Name System:* Sistema de Nombres de Dominio. Un servidor DNS almacena una base de datos con información sobre el nombre del dominio y la dirección IP a la que está vinculada. Al intentar conectar a `ddavo.me`, el cliente pregunta al servidor cual es la dirección IP asociada a esa dirección, y se conecta a tal IP, en este caso 37.152.88.18.

2.3.1.2 Transporte

Se encapsulan los datos de aplicación en un segmento o datagrama, dependiendo si el protocolo usado es TCP o UDP. Se encarga de transportar los datos por una red independientemente de la topología física.

- **TCP** *Transmission Control Protocol:* Protocolo de Control de Transmisión. Se aplica a los paquetes para administrarles un orden y un sistema de comprobación de errores. Con todas las funcionalidades, ocupa bastante espacio, lo que aumenta la latencia, aunque es más fiable.
- **UDP** *User Datagram Protocol:* Es un protocolo muy minimalista. A diferencia del TCP, no garantiza que los paquetes lleguen, o lleguen en orden, o protección ante duplicados. Reduce mucho la latencia ya que no usa *handshaking*.

2.3.1.3 Red

El objetivo es que los datos lleguen del origen al destino, aún cuando no están conectados directamente. Los enrutadores o *routers* son los dispositivos que cumplen esta función.

- **IP** *Internet Protocol*: Protocolo de Internet. Envía datagramas o paquetes de red a través de redes. Tiene función de enrutamiento que es la que permite la interconexión de redes, y la existencia de Internet. Encapsula el paquete definiendo en el *header* (cabecera) las direcciones IP del servidor y el cliente, o remitente y destinatario. La versión usada actualmente es IPv4, pero poco a poco se va abriendo paso la versión IPv6.
- **ICMP** *Internect Control Message Protocol*: Es un protocolo que no es usado por aplicaciones de usuario (a excepción de herramientas de diagnóstico como ping o traceroute). Lo usan los dispositivos de red para enviar notificaciones o mensajes de error indicando que un servicio no está disponible.

2.3.1.4 Link

Capa encargada del acceso al medio físico de la red. También cumple otras funciones como incluir una comprobación de errores e identificar cada dispositivo de forma única.

- **ARP** *Address Resolution Protocol:* Protocolo de resolución de direcciones. Es un protocolo que convierte direcciones de la capa de Red a la capa de Enlace (dir. IP a dir. MAC).
- **MAC** *Media access control*: Control de acceso al medio. Es un conjunto de protocolos (Como Ethernet o IEEE 802.11 [WiFi]) encargados de asignar el medio físico de la red. Evita colisiones entre paquetes asegurándose de que el medio está libre y evitando así la transmisión simultánea.

3. El simulador de redes

La parte práctica del Proyecto de Investigación, el simulador de redes de nombre *InvProy*, ha sido la parte más extensa del proyecto, que más tiempo, esfuerzo y recursos ha ocupado. Como curiosidad, el nombre de *InvProy* viene del acrónimo formado mediante la permutación de las palabras "Proyecto de Investigación", quedando *"Investigación ~~de~~ Proyecto"* y de ahí *InvProy*.

3.1 Herramientas usadas en la creación del software

Todo el software que se ha usado para la creación de este programa es software libre. A continuación, se listan algunas herramientas usadas en la creación del programa.

- **GNU/Linux** También llamado incorrectamente sólo Linux, es una manera de llamar al Sistema Operativo (OS) combinación del kernel Linux (Basado en Unix) y el OS GNU (Acrónimo recursivo *GNU's Not Unix*, o GNU no es Unix). Es el gran ejemplo por excelencia del Software Libre. Es el sistema operativo más utilizado, pues es usado en la mayoría de los servidores. Puedes instalar Linux desde el código fuente o instalar distribuciones o *distros* (conjunto de software preconfigurado y compilado que contiene GNU, Linux y otros paquetes). Se han usado Arch Linux, Ubuntu 16 y MaX 8.

- **Python** Es un lenguaje de programación interpretado (sólo se traduce el programa a código máquina cuando se debe ejecutar esa parte del código, por lo que no hace falta compilarlo) que destaca porque sus programas poseen una sintaxis más legible que la de el resto de lenguajes. Soporta tanto programación imperativa como programación orientada a objetos. Usa variables dinámicas, es multiplataforma, y, además, es de código abierto, lo que permite distribuir el programa en Windows al distribuir los binarios de Python junto a él. Se ha usado Python 3.4.

- **Gtk+** Es un conjunto de bibliotecas o librerías (conjunto de funciones y clases ya definidas preparadas para el uso de los programadores) desarrollado por la GNOME foundation destinado a la creación de Interfaces Gráficas de Usuario (GUI), también, al igual que Linux forma parte del proyecto GNU.

- **Git** Es un software diseñado por Linus Torvalds con el que se puede crear un Sistema de Control de Versiones (VCS). Este programa te permite de forma sencilla volver a una versión o *commit* anterior del programa, así como enviarlas a un repositorio remoto. Su punto fuerte son las *branches* o "ramificaciones"del código, haciendo que la rama *master* (principal) siempre sea funcional. Para ello creamos una nueva rama para cada nueva funcionalidad del programa.

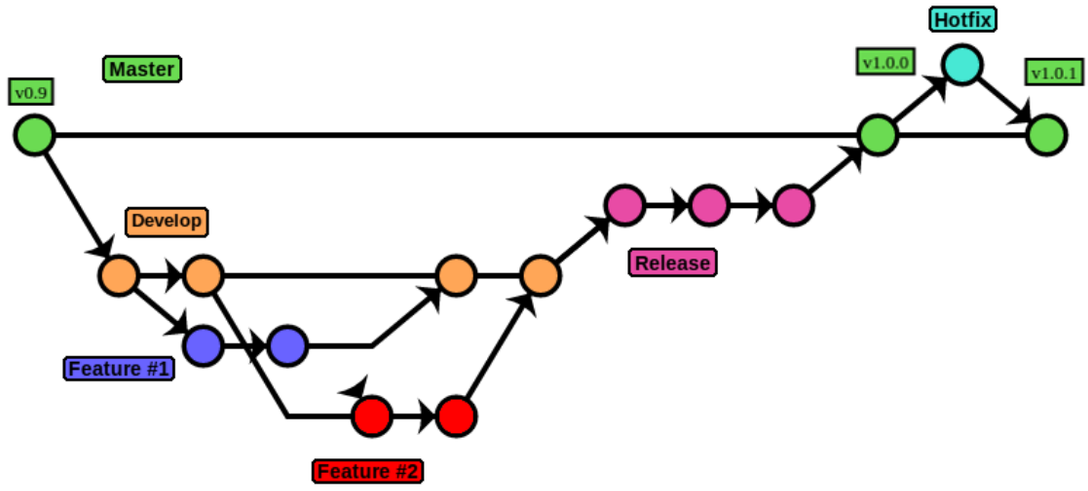

Figura 3.1: *Gitflow* o flujo de trabajo de Git

- **GitHub** es una plataforma de desarrollo colaborativo que te permite alojar tus repositorios Git. Su uso es gratuito si el código almacenado es público. Una de sus funciones estrella es la visualización online del repositorio, con la que cualquier persona tiene acceso al código y los archivos antes de descargarlos. Otra función útil es el apartado de *Issues*, en el que los usuarios de tu código pueden reportar los bugs del programa o aportar nuevas ideas en forma de "foro". Tanto el programa como este documento están disponibles en GitHub en los siguientes enlaces. `https://github.com/daviddavo/InvProy` y `https://github.com/daviddavo/InvProy-tex`

- **Wireshark** Wireshark es un *packet sniffer* o analizador de paquetes; te muestra los paquetes de red reales enviados y recibidos por una tarjeta de red, lo que facilita la creación del simulador de redes. También te separa las distintas partes de la encapsulación del paquete.

3.2 Instalación

3.2.1 Ubuntu / Debian

En caso de no estar en los repositorios, hay que hacerlo manualmente. Descargue el paquete de `https://github.com/daviddavo/InvProy/releases/latest`. Una vez descargado, abra una terminal donde se haya descargado el paquete e instálelo.

```
~ $ cd Descargas
Descargas $ sudo dpkg -i invproy*.deb
```

3.3 Uso del programa

Esta guía ha sido creada usando la versión v0.2.4-alpha, por lo que en algunos apartados pueden haberse realizado cambios en versiones posteriores.

El programa está siendo diseñado para tener una mayor facilidad de uso, pensando en una interfaz intuitiva y simple que pueda ser utilizada sin la necesidad de ningún apoyo externo al programa (instrucciones, documentación, tutor). Para ello, en una versión próxima será añadido un asistente o 'tutorial' que guíe a los alumnos la primera vez que se acceda al programa; además de añadir más información, dentro de la interfaz, sobre redes.

A continuación, se incluye una captura de pantalla de la interfaz de InvProy Alpha, explicando el funcionamiento de los distintos botones de la interfaz.

1-5. También se puede activar con las letras Q, W, E, R, T; respectivamente. Los botones te permiten (de izquierda a derecha): colocar un router, colocar un switch, conectar dos objetos, colocar un ordenador y colocar un hub.
6. Abre el menú de "Información de dispositivos", que proporciona información como la dirección IP y MAC, el nombre, o los dispositivos a los que está conectado. (Ver figura 3.3).
7. Te permite enviar un ping de un ordenador a otro sin necesidad de conocer ambas direcciones IP (El botón funciona a partir de v0.3).
8. Abre el menú de archivo, en el que puedes abrir, guardar y crear un archivo.
9. Es la ventana donde puedes colocar los objetos. Puedes moverte a través de ella y en el menú de 'Ver' puedes cambiar el que se vea la rejilla de fondo. La extensión de está puede cambiarse.
10. Aquí se encuentra una barra con información sobre el funcionamiento actual del programa.

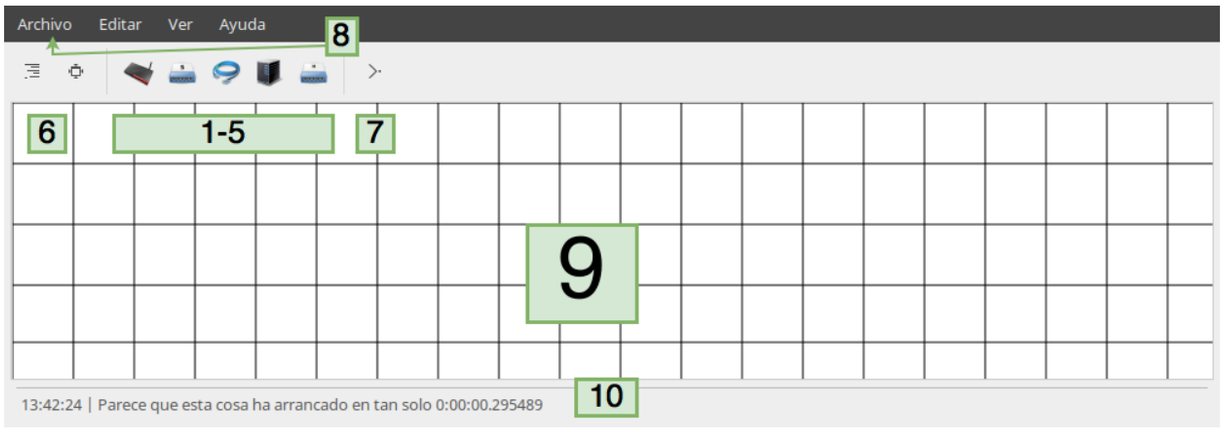

Figura 3.2: Interfaz de InvProy Alpha. Al usar Gtk+, los temas se pueden cambiar, así que la apariencia del programa puede ser distinta dependiendo del tema de escritorio que estés usando. La ventana es redimensionable, en este caso tiene una proporción que permita explicar la interfaz sin usar demasiado espacio vertical.

Para incluir un objeto en la rejilla, se hace click en el icono del objeto y luego en el lugar donde se quiera poner. Cuando tenemos dos objetos podemos conectarlos si hacemos click primero en el icono del cable, luego en un objeto y después en otro.

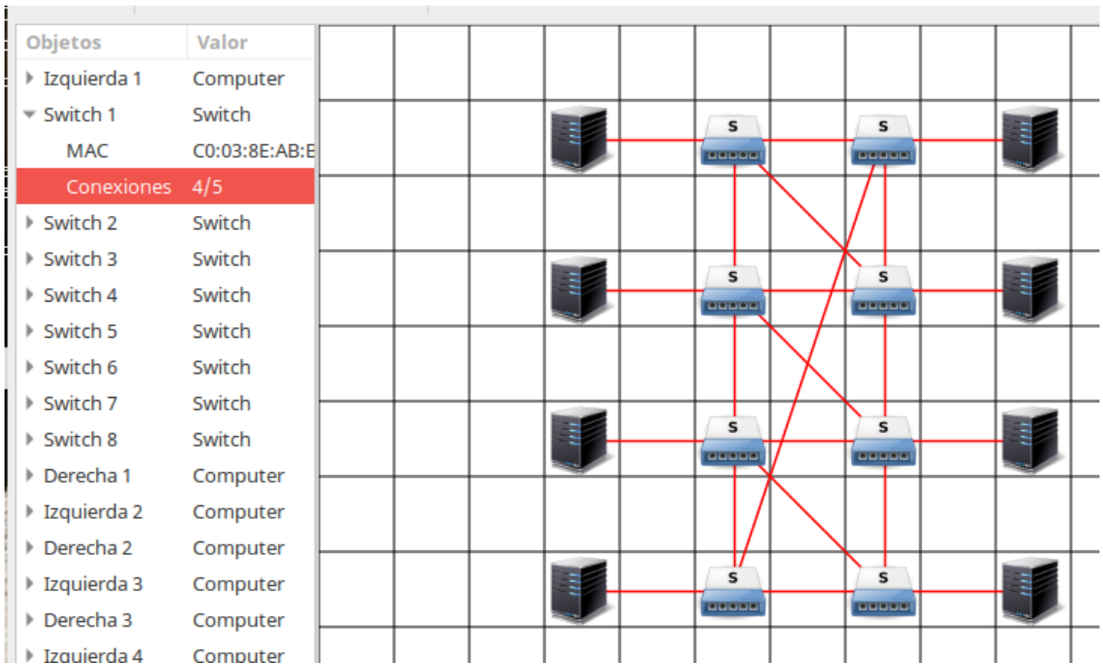

Figura 3.3: Menú de **Información de Dispositivos** junto a una red de topología de malla

3.4 Funcionamiento del programa

El programa posee distintas clases. Se pueden diferenciar en cuatro tipos: Clases de Interfaz (`MainClase`, `w_changethings`...), Clases de Dispositivos (`ObjetoBase, Switch, Computador`), Clases de Red (`packet`, `frame`) y clases de apoyo (`MAC`, `IP`, `Port`, `Cable`).

3.4.1 Main.py

Es el archivo principal del programa. Contiene las funciones más importantes, además de las clases para crear los objetos.

3.4.1.1 MainClase

Es la clase principal de la interfaz del programa. Se encarga de administrar la ventana principal de la interfaz. Posee varias funciones como `on_key_press_event` y `on_key_release_event` , que actúan cada vez que se pulsa una tecla y se encargan de hacer las acciones necesarias para esa tecla (o combinación de teclas). Otra función importante es `toolbutton_clicked`, que se acciona cada vez que se pulsa un botón de la botonera. También contiene una subclase, `ObjLst` , encargada de la lista de objetos de la parte izquierda de la interfaz.

3.4.1.2 Grid

Es la clase de la rejilla. Tiene varias 'capas', una para los cables, otra para el fondo, otra para los dispositivos... Así, en el caso de que dos elementos se solapen, los dispositivos siempre permanecerán al frente del fondo y los cables. El fondo se hace creando una línea horizontal cada `sq` píxeles, y otras tantas verticales del mismo modo, siendo `sq` el parámetro `viewport-sqres` del archivo `Config.ini`.

`clicked_on_grid`: Función que se encarga de realizar distintas acciones dependiendo de dónde se haya hecho click dentro de la rejilla.

`gridparser`: Convierte coordenadas de la rejilla, a coordenadas en píxeles (usadas por Gtk).

`resizetogrid`: Dada una imagen, la convierte al tamaño de un cuadrado de la rejilla.

`searchforobject`: Dadas unas coordenadas, comprueba si hay un objeto en estas.

`moveto`: Mueve una imagen dada a unas coordenadas dadas.

3.4.1.3 ObjetoBase

En Python, existe la herencia de clases. Esto quiere decir que una clase puede heredar las funciones y los atributos de otra, en forma de cascada. La clase principal de la que heredan el resto de dispositivos de red es `ObjetoBase`. Algunas de sus funciones son estas:

- `compcon`: Es una función que poseen todos los dispositivos de red, que dado un objeto `Computador`, retorna todos los ordenadores que están conectados a la misma red. Está formada por una lista que contendrá los ordenadores conectados y una función llamada subcompcon, que comprueba las conexiones del objeto, añadiendo la conexión a la lista si es un ordenador. En el caso de que sea un conmutador o un concentrador, llama a la función `subcompcon` con ese objeto como argumento, por lo que comprueba las conexiones de ese objeto y las añade a la lista del primero, entrando en un bucle hasta que ha comprobado toda la red. La función es usada por el programa cuando es necesario comprobar si dos ordenadores están conectados, por ejemplo. (Ver figura 3.4)
- `load`: Al cargar un objeto de un archivo, hay determinadas propiedades del objeto que deben ser establecidas de cero, y determinadas funciones que deben ser llamadas, para ello existe esta función.
- `update`: Esta función, bastante importante se encarga de actualizar la información del objeto en la interfaz de usuario. Es llamada cada vez que se produce un cambio en el objeto; como al conectarlo, editar el nombre, o desconectarlo de otros objetos.

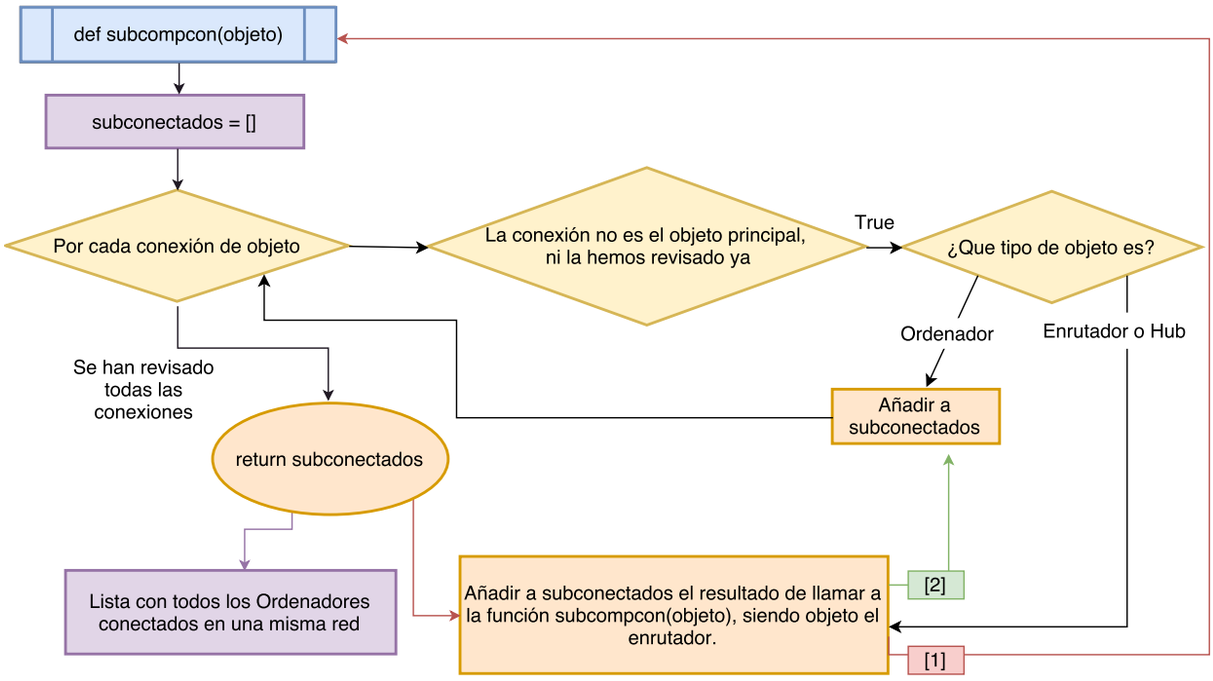

Figura 3.4: Diagrama de flujo del funcionamiento de la función `compcon`.

- `connect`: Esta función se encarga de establecer las conexiones entre dos objetos.
- `packet_received`: Función ejecutada cuando un dispositivo recibe un paquete.

3.4.1.4 Port y w_switch_table

`Port` es una clase que usan los conmutadores y concentradores. Simula un puerto de red. Tan sólo posee tres funciones: `connect`, para conectar un objeto al puerto; `disconnect`, para desconectarlo y `is_available`, para saber si el puerto está disponible u ocupado.

La clase `w_switch_table` es la encargada de la ventana de visualización de la tabla de enrutamiento del Switch.

3.4.1.5 Clases de paquetes de red

Ocupan entre el 25 % y el 30 % del código. Son clases como `packet` (la clase base), `eth` (paquete con *frame* aplicado, `icmp` (paquete con ICMP) y la última clase `Ping`, que hereda de `icmp` y se encarga de crear un paquete de red, bit a bit, dados una dirección IP de destino y de origen. Entre todas estas clases debemos destacar dos funciones:

- `animate` es una función que poseen todos los tipos de paquetes de red y se encarga de poner un paquete de red en la interfaz, y de que este se mueva. Para ello, hace una combinación de dos movimientos, uno en el eje x y otro en el eje y, la longitud que debe de moverse en total la divide entre el número total de fotogramas y así consigue la distancia que debe moverse

cada fotograma. Cuenta dentro con una subfunción, `iteration`, que se encarga de poner la imagen cada fotograma en su sitio y eliminar la imagen del fotograma anterior.

- `create` es una función propiedad de `Ping`, que dadas una dirección IP de destino y origen, crea un paquete de red, bit a bit, basado en el modelo real de paquetes de Ping inspeccionado por Wireshark. Es una función que, aunque parezca sencilla, fueron bastantes horas de trabajo; pues es bastante complejo tratar con bits.

```
1676    def create(r, sourceip, desti_ip, *n, payload=int( 4.3*10**19 ) << 6 | 42, \
1677        flags=0b010, ttl=32):
1678        self = Ping()
1679        if r == 0:
1680            Type = 8
1681            self.color = "#4CAF50"
1682        if r == 1:
1683            Type = 0
1684            self.color = "#F44336"
1685
1686        self.payload = payload
1687
1688        vihltos   = 0b0100010100000000
1689        #20 Ipheader + 8 ICMPHEader + Payload
1690        lenght    = int( 20 + 8 + ( int(math.log(payload, 2))+1)/8 ) #In Bytes
1691        frag_off  = 0b0000000000000
1692        protocol  = 1
1693        checksum  = 0 #No es necesario porque no hay cables
1694        sourceip  = int(sourceip)
1695        desti_ip  = int(desti_ip)
1696        identific = Ping.identifi
1697        Ping.identifi += 1
1698
1699        self.ip_header = ((((((((vihltos << 16 | lenght)<<16 | identific) << 3 | flags) << 13 | frag_off) \
1700        << 8 | ttl) << 8 | protocol) << 16 | checksum) << 32 | sourceip) << 32 | desti_ip)
1701
1702        identifier = 1*2**15 + 42 * 2**8 + 42
1703        Code = 0
1704        self.icmp_header = ((((((((Type << 8) | Code)<< 16) | checksum) << 16) | identifier) << 16) | identific)
1705        self.pck = icmp(self.ip_header, self.icmp_header, self.payload)
1706
1707        self.str = self.pck.str
1708        self.lenght = self.pck.lenght
1709        self.bits = self.pck.bits
1710
1711        return self
```

3.4.2 Dispositivos

Existen cuatro tipos dispositivos: los ordenadores, que tienen la mayor programación; los Switches, que se encargan de manejar los paquetes de red; los Hubs, que son como los Switches, pero reenvían los paquetes por todos sus puertos y los Routers, que tan sólo existen de forma visual, pero no tienen ninguna función de momento. Por lo que sólo vamos a hablar de los Switches y los Ordenadores. Para cambiar los parámetros hay que hacer click derecho en el dispositivo al que se le deseen cambiar los parámetros y luego en la entrada de 'Editar objeto'. A lo que aparecerá una ventana como la de Fig. A.5 en la que se podrán cambiar parámetros como el nombre, la dirección MAC o la dirección IP.

Los ordenadores tienen una función especial que es la de crear y enviar los paquetes de red. Para ello, en el menú emergente que aparece al hacer click derecho en el objeto, hacemos click en la entrada de 'Ping'. Para que el paquete llegue al otro computador, ambos deben tener una dirección IP, y estar conectados a la misma red. Se introduce la dirección IP del dispositivo y se pulsa en el botón de 'Ping!' (Ver Fig. A.2 y Fig. A.3). A continuación veremos el paquete de red buscando su objetivo, la primera vez no irá directamente, ya que los Switches están aún aprendiendo el camino, pero el paquete de vuelta y todos los siguientes paquetes seguirán la misma ruta (Ver Fig. A.6). El ordenador crea un paquete de red usando los protocolos de Ethernet (IEEE 802.11), TCP, IPv4 e ICMP. La función que se encarga de esto es `create`.

Los Switches se encargan de redireccionar los paquetes de red. La primera vez que les llega un paquete, al no saber la ubicación física del destino, siguen este algoritmo:

```
1181        if int(macd, 2) in dic and ttl > 0:
1182            pck.animate(self, dic[int(macd, 2)])
1183
1184        elif int(macd, 2) in [x[0] for x in self.table] and ttl >= 0:
1185            for x in self.table:
1186                if x[0] == int(macd, 2):
1187                    pck.animate(self, self.pdic[x[1]])
1188
1189        elif "Switch" in [x.objectype for x in self.connections] and ttl >= 0:
1190            logger.debug("Ahora lo enviamos al siguiente router")
1191            tmplst = self.connections[:] #Crea una nueva copia de la lista
1192            logger.debug(tmplst)
1193            for i in tmplst:
1194                if int(macs, 2) == int(i.macdir):
1195                    logger.debug("REMOVING %s", i)
1196                    tmplst.remove(i)
1197            try:
1198                tmplst.remove(*[x for x in tmplst if x.objectype == "Computer"])
1199            except TypeError:
1200                pass
1201            logger.debug("Tmplst: %s", tmplst)
1202            obj = choice(tmplst)
1203            logger.debug("Sending to: {}".format( obj))
1204            pck.animate(self, obj)
```

Este algoritmo esta basado en el que usan los conmutadores reales y, traducido a lenguaje humano, vendría a ser:

Si la dirección MAC de destino del paquete recibido se encuentra directamente conectado al Switch y el TTL del paquete es mayor que cero: Enviar el paquete a ese dispositivo.

Al no cumplirse la condición anterior, si el paquete se encuentra en mi tabla de enrutamiento y el TTL del paquete es mayor que cero: Enviamos el paquete por el puerto al que está asignada la dirección MAC en la tabla.

Al no cumplirse las condiciones anteriores, si hay un Switch en mis conexiones y el TTL del paquete es mayor a 0: Enviar el paquete a uno de los Switches de forma aleatoria.

Cuando recibe un paquete, también añade a la *Routing Table* o tabla de enrutación una entrada con la dirección MAC del remitente del paquete y el puerto por el que ha llegado, así cuando le llegue un paquete el router conocerá el puerto por el que enviarlo.

```
1142            for tab in self.table:
1143                if tab[2] <= time.time():
1144                    logger.debug("Ha llegado tu hora")
1145                    self.table.remove(tab)
1146                    self.wtable.remove(tab)
1147                if tab[0] == int(macd, 2):
1148                    logger.debug("TAB[0] == mcd")
1149                    tab[2] = int(time.time()+self.timeout)
1150                    for row in self.wtable.store:
1151                        logger.debug("%s, %s", row[0], tab[0])
1152                        if int(row[0].replace(":", ""), 16) == tab[0]:
1153                            row[3] = int(time.time()+self.timeout)
1154            if int(macs, 2) not in [x[0] for x in self.table]:
1155                tmp = [int(macs, 2), port, int(time.time()+self.timeout)]
1156                self.table.append(tmp)
1157                tmp = [readmac, port, int(time.time()+self.timeout)]
1158                self.wtable.append(tmp)
```

Este es el código que cumple esta función. Cada elemento en la tabla tiene un tiempo establecido en el que caduca la entrada. Lo que hace esta parte del código es comprobar si este tiempo ha caducado, actualizar la fecha de caducidad si la dirección MAC ya está en la tabla o añadirlo de nuevo en la tabla si la dirección no está.

3.5 Versión actual del programa (0.2.4-alpha)

En la **versión 0.1** se introdujo toda la interfaz, las conexiones, los dispositivos... Pero aún no se podían enviar ni recibir paquetes de red. En la **versión 0.2** se introdujo esta posibilidad, junto a otras cosas como el enrutamiento de paquetes. El programa es considerado una versión *alpha*, ya que aún está en desarrollo y no es un programa terminado.

El programa te permite, por el momento, hacer una simulación de red simple. Se podría decir que es una base sobre la que se pueden ir añadiendo más funcionalidades, como el soporte para otros protocolos, o un modo 'explicatorio' que enseñe a los alumnos lo que está pasando en la red. En la **versión 0.2.3-alpha** del programa se introdujo el "Ping", es decir, la posibilidad de enviar un paquete de prueba a otro dispositivo de la misma red. También se han introducido algunos cambios en la interfaz, uno de ellos, bastante útil para el aprendizaje: los cuadros de texto en los que se introducen direcciones IP, cambian de color entre rojo, naranja o verde, dependiendo si la IP introducida no es válida, está incompleta o es válida, respectivamente.

En la **última actualización**, la `0.2.4-alpha` de Enero de 2017 se han producido muchas mejoras en el código. De limpieza, optimización y adecuación a estándares (PEP 8 [5]), aprovechando los nuevos conocimientos adquiridos. Al final del Apéndice A se encuentran varios ejemplos.

3.6 Desarrollo del proyecto

En cuanto al código, a pesar de la gran extensión del programa, han sido escritas muchas más líneas, que han sido en algún momento eliminadas o reemplazadas. El desarrollo del proyecto puede dividirse en 4 fases a lo largo de 3 trimestres y la actualidad.

En la primera fase, de Noviembre a Febrero/Marzo del curso 2015/2016 he ido aprendiendo sobre todo de Gtk+, la librería para la interfaz del programa. Al empezar el proyecto mis conocimientos sobre esta librería eran nulos; y sobre Python, el lenguaje de programación, eran demasiado básicos. También aprendí bastante sobre redes informáticas.

En la segunda fase, se fue desarrollando la "base"del programa, transcurre de Febrero-Marzo a finales de curso. La interfaz, las ideas, las conexiones de los cables... Se construye la versión 0.1, como menciono en 3.5. El programa contaba con unas 700 líneas en `Main.py`

En la tercera fase se desarrolla la gran parte del programa, aquí es cuando llega a las 2000 líneas, sin mencionar los pequeños módulos y otros archivos. Transcurre en verano, entre Junio y mediados de Agosto. Con una media de 200-300 líneas semanales y picos de hasta mil líneas entre el 7 y el 14 de Agosto, ha sido posible cumplir el objetivo de crear un pequeño simulador de redes.

La cuarta fase es la del desarrollo posterior del proyecto. A partir de Octubre, cuando se presenta como Proyecto de Investigación de Bachillerato de Excelencia en el IES Palas Atenea, se sigue desarrollando el programa con los nuevos conocimientos adquiridos, siendo presentado a concursos y con el objetivo de lanzar la versión 1.0-β antes de Septiembre de 2017.

He notado bastante la adquisición de experiencia, ya que tardé prácticamente 5-6 meses en hacer las primeras 500 líneas; pero en verano, conforme iba programando más, conseguí llegar a hacer más de 1000 líneas en una semana. También, al leer el código antiguo se notan bastantes errores debidos a la falta de experiencia, que deben ser corregidos si aún no se ha hecho (Ver Fig. A.7)

3.6.1 Obstaculos en el desarrollo del proyecto

Durante el desarrollo del proyecto han surgido bastantes trabas y contratiempos, que he conseguido solucionar. Muchos de ellos surgen por la falta del gran conocimiento técnico necesario para la creación de un software tan específico, han sido muchas horas de mirar la documentación de las librerías [6], y pedir ayuda por foros para intentar solucionar dudas y bugs.

En algunas ocasiones no han sido errores, sino falta de conocimiento para el desarrollo de determinadas funciones lo que ha creado pausas de hasta dos semanas en la acción de escribir el programa. Gracias a comunidades como ***stackoverflow*** he conseguido solucionar muchas de las dudas y errores básicos del programa.

3.7 Conclusión

Lo más difícil fue empezar. El tratar de aprender tanta información de golpe de forma autodidacta. Aunque ya supiese un poco sobre programación en Python, no tenía casi experiencia, aprender a usar la librería de Gtk+, aprender sobre redes, aprender sobre un uso más extenso de GNU/Linux, aprender sobre LaTeX, etc. fue bastante cansado. pero eso es lo mejor, todo lo que he aprendido y, sobre todo, la experiencia que he adquirido en el campo de la programación.

A la hora de programar, al principio el ritmo era muy lento, de unas 200 líneas al mes, con pausas de semanas para solucionar problemas y errores. Poco a poco se fue acelerando hasta llegar a finales de Julio, donde hacía más de 100 líneas diarias. Aprendiendo en el proceso el uso de clases y decoradores.

Pese a que es verdad que falta incluir más protocolos y algunas funcionalidades bastante básicas (como mover objetos), estoy bastante satisfecho con la versión actual del programa, que se ha realizado con bastante poco tiempo, ya que tiene las bases, y creo que añadir un nuevo protocolo, o una nueva funcionalidad no serían más que unas horas delante de la pantalla y el teclado del ordenador.

Resumiendo:

- Se ha creado un simulador de redes escrito en Python que demuestra el uso de Ping's y enrutamiento de paquetes
- Con el programa actual, es fácil añadir nuevos paquetes, dispositivos o funcionalidades, entre otras cosas
- Se ha desarrollado íntegramente con software libre
- Tiene uso didáctico en 4º de la ESO, 1º y 2º de Bachillerato, pues se incluye 'Redes informáticas' en el temario de Informática y TICO, además de 'Programación'
- Los alumnos pueden modificar y ampliar el programa, para toda la comunidad educativa, en GitHub

A. Imágenes y capturas del programa

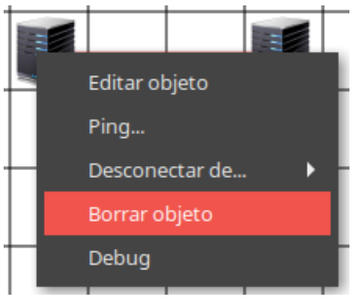

Figura A.1: Captura: Click derecho en un computador

Figura A.2: Captura: Ventana para enviar ping. Está en rojo porque la IP introducida no es válida.

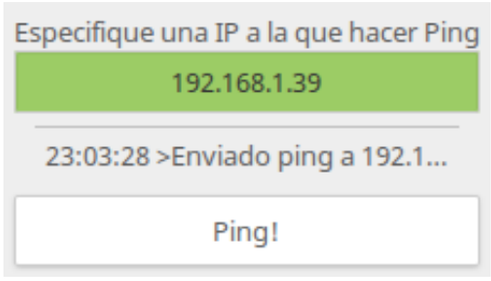

Figura A.3: Captura: Igual que A.2, pero con una IP válida.

Figura A.4: Captura: Ventana con la tabla que poseé el Switch.

Figura A.5: Captura: Ventana de edición de propiedades de objeto.

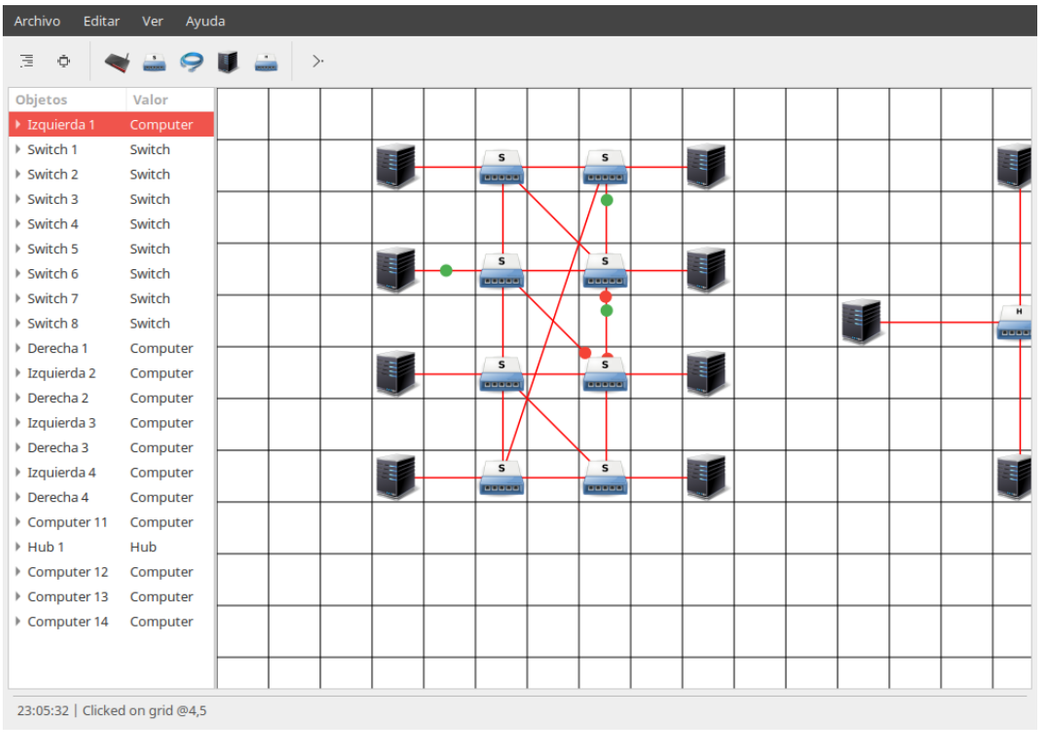

Figura A.6: Captura: Paquetes viajando por una red de ejemplo.

Antes (0.2.3):

```python
#Comprueba si el objeto tiene una ip asignada
def has_ip(self, objeto):
    try:
        if objeto.IP != None:
            return True
        else:
            return False
    except:
        return False
```

```python
#Comprueba si un objeto está conectado a otro.
def isconnected(self, objeto):
    cons = compcon(self)
    if objeto in cons:
        return True
    else:
        return False
```

```python
def searchforobject(self, x, y):
        global allobjects
        localvar = False
        for i in range(len(allobjects)):
            if allobjects[i].x == x:
                if allobjects[i].y == y:
                    localvar = True
                    objeto = allobjects[i]
                    break
        if localvar == True:
            return objeto
        else:
            return False
```

Después (0.2.4):

```python
@staticmethod
def has_ip(objeto):
    """Comprueba si el objeto tiene una ip asignada"""
    return bool(objeto.IP != None)
```

```python
def isconnected(self, objeto):
    """Comprueba si un objeto está conectado a otro."""
    return bool(objeto in compcon(self))
```

```python
def searchforobject(self, x, y):
        for i in allobjects:
            if i.x == x and i.y == y: return i
        return False
```

Figura A.7: Código: Algunas diferencias entre versiones. Observamos la reducción del código.

Bibliografía

[1] Cisco. (2015). Cisco visual networking index: Forecast and methodology, dirección: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

[2] Free Software Foundation. (2013). Filosofía del proyecto GNU, dirección: https://www.gnu.org/philosophy/philosophy.html.

[3] R. Stallman. (2014). Charla: Free software, free society: R. Stallman at TEDxGeneva, dirección: https://www.youtube.com/watch?v=Ag1AKIl_2GM.

[4] ——, (2013). Conferencia sobre software libre en la universidad de Jaume I, dirección: https://www.youtube.com/watch?v=5t_EcPTEzh4.

[5] N. C. Guido van Rossum Barry Warsaw. (2013). Style guide for python code, dirección: https://www.python.org/dev/peps/pep-0008/.

[6] C. Reiter (lazka en GitHub). (2016). Python GObject Introspection API Reference, dirección: https://lazka.github.io/pgi-docs/.

[7] Python Software Foundation. (2016). What is Python? executive summary, dirección: https://www.python.org/doc/essays/blurb/.

[8] Real Academia Española, *Diccionario de la lengua española, ed. XXIII*. 2014.

[9] BICSI, *Network Design Basics for Cabling Professionals*. 2002.

[10] R. Braden, *Request for Comments 1122*, 1989.

[11] Alumnado de la asignatura de Software Libre del Máster en Sistemas Telemáticos e Informáticos de la Universidad Rey Juan Carlos. (2013). Traducción de la licencia GPLv3 al español, dirección: https://lslspanish.github.io/translation_GPLv3_to_spanish/.

[12] University of Cambridge Computer Laboratory. (2001). A brief informal history of the computer laboratory, dirección: https://www.cl.cam.ac.uk/events/EDSAC99/history.html.

[13] Microsoft Developer Network. (2015). Serialización, dirección: https://msdn.microsoft.com/es-es/library/ms233843.aspx?f=255&MSPPError=-2147217396.

[14] Wikipedia. (2016). Local area network: History, dirección: https://en.wikipedia.org/wiki/Local_area_network#History.

[15] All About Circuits. (2016). Introduction to boolean algebra, dirección: http://www.allaboutcircuits.com/textbook/digital/chpt-7/introduction-boolean-algebra/.

[16] M. Kanat-Alexander, *Code simplicity*. 2012.

[17] D. Davó. (2016). Invproy α, dirección: https://github.com/daviddavo/InvProy-tex/raw/master/InvProy.pdf.