Sistema de comunicación local mediante Walkie-Talkies TCP/IP

Diseño del servidor en una Raspberry Pi Zero 2W

David Gómez Pérez, 1563212

8 de julio de 2024

Resumen– Queremos desarrollar un sistema de comunicaciones local basado en unos walkie-talkies. Pero en lugar de usar unos walkie-talkies convencionales basados en radiofrecuencia usaremos unos que usan la red WiFi para comunicarse mediante el protocolo TCP/IP. De esta forma tendremos un mayor control y capacidad de monitoreo de las comunicaciones. Para alojar la red local así como la aplicación de control se usara una Raspberry Pi Zero 2W que actuará de servidor. En este documento nos centraremos en el desarrollo del servidor.

Palabras clave– Walkie-Talkie, Red Local, Raspberry, TCP/IP, Control, Comunicación, Servidor

Abstract– We want to develop a local communications system based on walkie-talkies. But instead of using conventional walkie-talkies based on radio frequency we will use ones that use the WiFi network to communicate through the TCP/IP protocol. In this way we will have a greater control and monitoring capacity of the communications. To host the local network as well as the control application we will use a Raspberry Pi Zero 2W that will act as a server.
In this document we will focus on the development of the server.

Keywords– Walkie-Talkie, Local Network, Raspberry, TCP/IP, Control, Communication, Server

1 Introducción - Objetivos

Vamos a diseñar e implementar el servidor de nuestra red de comunicación local usando para ello una Raspberry Pi Zero 2W. Para que nuestro sistema de comunicación funcione lo primero que necesitamos es disponer de una red WLAN a la que conectar los walkie-talkies así que veremos como usar nuestra Raspberry Pi Zero 2W para alojar una red local. Los motivos para alojar nuestra propia red local y no usar un WiFi existente son principalmente no depender de la existencia de una red WiFi y simplificar el diseño de los walkie-talkies ya que no existirá necesidad de que el usuario pueda conectarlos a una red, las credenciales de nuestra red local (SSID y contraseña) pueden ir en un archivo de configuración del walkie. A su vez las IPs se asignan de forma automática al conectarse a una red así que sería difícil que un walkie conozca la IP de cada uno de los otros walkies pero es muy sencillo conocer la del host de la red así que haremos que todos los walkies manden sus mensajes a la Raspberry y esta se encargará de reenviarlos a los demás dependiendo de la configuración elegida por el usuario.

Una vez tengamos la red operativa deberemos encontrar el modo de controlar las comunicaciones, queremos poder:

- Silenciar dispositivos
- Ensordecer dispositivos
- Comunicaciones independientes

Para ello aprovechando que Raspberry OS cuenta con el kernel de Linux [1] usaremos las IPtables para decidir los mensajes de que dispositivos se reenvían o a que dispositivos no hay que reenviarles los mensajes. La razón de usar IPtables es principalmente la poca memoria RAM disponible en la Raspberry Zero. Debido a esto hemos decidido abstraer toda la gestión de los paquetes de la aplicación, así que aunque la Raspberry actúe de servidor, nuestra aplicación no será un servidor per se. Realmente se trata de una

• E-mail de contacte: davidgope65@gmail.com
• Menció realitzada: Enginyeria de Computadors
• Treball tutoritzat per: Dolores Isabel Rexachs del Rosario (CAOS)
• Curs 2023/24

aplicación de administración del sistema que se encarga de interactuar con el Sistema Operativo para modificar las reglas del controlador de red.

Por último necesitaremos una aplicación amigable con el usuario que permita el control de las comunicaciones sin tener conocimientos de IPtables, para ello diseñaremos una interfaz gráfica fácil y sencilla y dotaremos a la aplicación de alguna interfaz con el SO para conocer el estado actual de las tablas y poder modificarlo.

Con el objetivo global de implementar un sistema de comunicación local fiable y controlado en mente podemos definir los siguientes objetivos para este documento:

- Usar una Raspberry Pi Zero 2W como host de una WLAN.
- Usar las IPtables del kernel de linux para implementar un proxy en las comunicaciones que permita a los walkie-talkies comunicarse.
- Desarrollar una aplicación con interfaz gráfica user-friendly que permita la gestión de los diferentes canales de comunicación. (Python-Tkinter)
- Dotar a la aplicación anterior de interacción con el SO para añadir/eliminar reglas de las IPtables.

2 Metodología

2.1. Proyecto global

Para el desarrollo del proyecto global seguiremos un proceso en V como el visto en (Fig.1) donde los requisitos y la arquitectura del sistema es trabajo realizado previamente y se consideran premisas para este trabajo como el uso de una Raspberry, la estructura cliente-servidor...

En este documento nos centraremos en el desarrollo del servidor TCP/IP y veremos el test del sistema en las conclusiones.

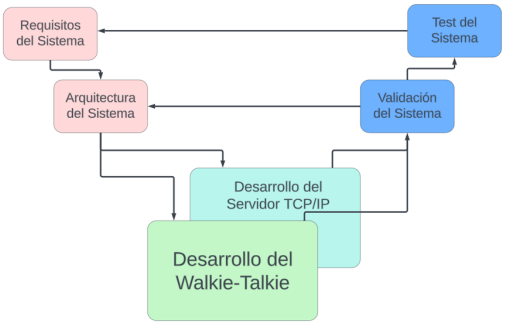

Fig. 1: Proceso de Desarrollo del Sistema de Comunicación Local

2.2. Servidor TCP/IP

Para el desarrollo del servidor se usará a su vez otro proceso en V (Fig.2) en el que primero se analizaran los requisitos de nuestro servidor, a continuación se diseñará el Front-End para definir así todas las posibilidades de interacción que ha de tener el usuario. A raíz de estas posibilidades ofrecidas al usuario debemos definir unas necesidades de control sobre la red y que información debe recibir el

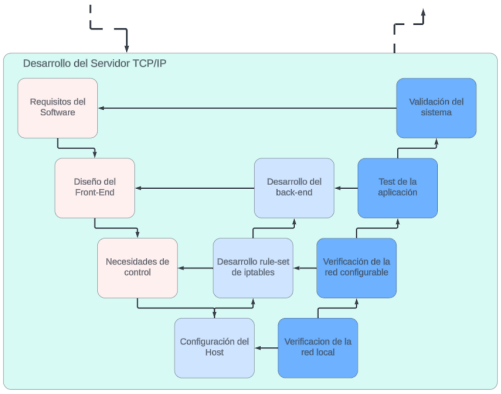

Fig. 2: Proceso de Desarrollo del Servidor TCP/IP

usuario. Una vez definidas las necesidades del servidor, empezaremos su implementación de mas bajo a mas alto nivel. Empezado por configurar nuestra Raspeberry como host de la red y verificar que se pueden conectar dispositivos a ella, luego se desarrollará y se verificará un rule-set para las IPtables el cual satisfaga todas las necesidades de control. Por ultimo se desarrollará el Back-End de nuestra aplicación de control para poder activar y desactivar reglas del rule-set desde la interfaz de usuario diseñada anteriormente. Ahora ya podemos validar nuestro servidor y comprobar que cumple todos los requisitos.

3 Contexto - Estado del Arte - Definiciones Generales

3.1. Contexto

El proyecto se enmarca en la necesidad de mejorar y modernizar los sistemas de comunicación local, aprovechando las ventajas de las tecnologías digitales. Tradicionalmente, los walkie-talkies han sido herramientas esenciales para la comunicación en diversas áreas, desde la seguridad hasta la coordinación de eventos y actividades al aire libre. Sin embargo, estos dispositivos presentan limitaciones en términos de alcance y funcionalidades.

Este proyecto fue pensado principalmente para ser usado en *Scape Rooms*, debido a la necesidad de que dos equipos pudiesen comunicarse solo cuando el *Game Master* quisiera que pudieran hacerlo, o bien que el *Game Master* quiera comunicar algo a un equipo sin que el otro lo escuche.

En este contexto, la propuesta del proyecto busca implementar una solución que combine la simplicidad y eficacia de los walkie-talkies con las capacidades avanzadas de las redes TCP/IP. El uso de una Raspberry Pi Zero 2W como servidor, la integración de IPtables para el control del tráfico de red y una aplicación servidor TCP son elementos clave en esta propuesta. La idea es proporcionar un sistema que permita una comunicación más versátil y controlada, facilitando la gestión de múltiples dispositivos y canales de comunicación.

3.2. Protocolo TCP/IP

El protocolo TCP/IP, siglas de Transmission Control Protocol/Internet Protocol, es un conjunto integral de protocolos de comunicaciones diseñado para permitir la intercone-

xión y el intercambio de datos entre sistemas de cómputo diversos a través de redes.[6] Este modelo de protocolo, fundamental para el funcionamiento de Internet, está estructurado en cuatro capas jerárquicas, cada una con funciones específicas:

- Capa de enlace de datos: Gestiona la transmisión física de datos y las conexiones entre dispositivos de red. Esta capa incluye protocolos como Ethernet y Wi-Fi, que definen cómo se estructuran y transmiten los datos a nivel de hardware.
- Capa de red: En esta capa opera el protocolo IP (Internet Protocol), que se encarga de la dirección y el enrutamiento de los paquetes de datos. IP asigna direcciones únicas a cada dispositivo en la red y determina la ruta óptima para que los datos lleguen a su destino. Existen dos versiones de IP en uso: IPv4, con direcciones de 32 bits, e IPv6, con direcciones de 128 bits, que permiten un número significativamente mayor de direcciones únicas.
- Capa de transporte: Incluye el protocolo TCP (Transmission Control Protocol), que proporciona una comunicación confiable y orientada a la conexión. TCP divide los datos en segmentos, los transmite y asegura que se reciban correctamente y en el orden adecuado.

Cada capa del modelo TCP/IP interactúa con las capas adyacentes, proporcionando una estructura modular que facilita la implementación, el mantenimiento y la evolución de redes de comunicación complejas. Este diseño modular permite que TCP/IP sea altamente escalable y adaptable, lo que ha contribuido a su adopción universal como la base de Internet y muchas otras redes de datos.

3.3. Raspberry Pi Zero 2W

La Raspberry Pi Zero 2W es una computadora de placa única, compacta y de bajo costo, diseñada y desarrollada por la Fundación Raspberry Pi[2]. Con su pequeño tamaño y capacidad de procesamiento, se ha convertido en una herramienta versátil para diversos proyectos de electrónica y computación. La Raspberry Pi Zero 2W puede ser configurada para servir diversos propósitos dentro de una red local como servidor web, servidor de archivos, homeassistant o como en nuestro caso que actuará de host de la red, de proxy y de servidor de control en las comunicaciones.

3.4. Python

Python es un lenguaje de programación interpretado, de alto nivel y de propósito general. Python se destaca por su sintaxis clara y legible, lo que facilita la escritura y el mantenimiento del código. Python se destaca por su capacidad de interactuar fácilmente con el sistema operativo (SO). Esta interacción es facilitada por una variedad de módulos y bibliotecas que permiten a los desarrolladores realizar tareas de administración del sistema, manipulación de archivos, ejecución de comandos del sistema y más.

3.4.1. Tkinter

Tkinter [4] es la biblioteca estándar de Python para la creación de interfaces gráficas de usuario (GUI). Proporciona una manera sencilla de crear ventanas, diálogos y otros elementos gráficos en aplicaciones de escritorio. Es una biblioteca fácil de usar, multiplataforma y que cuenta con una amplia documentación y comunidad.

3.4.2. Socket

El uso de sockets TCP en Python se refiere a la implementación de comunicación en red utilizando el protocolo de Control de Transmisión (TCP) mediante la biblioteca estándar socket. Un socket es un punto final para enviar y recibir datos a través de una red. Los sockets TCP son confiables y orientados a la conexión, lo que significa que garantizan la entrega de datos en el orden correcto y sin pérdida. El proceso general a seguir para usar sockets es el siguiente:

1. Creación del Socket: Se crea un objeto de socket TCP utilizando la función socket.socket() con los parámetros socket.AF_INET (para IPv4) y socket.SOCK_STREAM (para TCP).

2. Conexión: En el lado del cliente, se utiliza connect() para establecer una conexión con el servidor. En el lado del servidor, se utiliza bind() para asociar el socket a una dirección y puerto, listen() para esperar conexiones entrantes y accept() para aceptar una conexión.

3. Transmisión de Datos: Se utilizan send() y recv() para enviar y recibir datos respectivamente.

4. Cierre del Socket: Se cierra la conexión con close() una vez que la comunicación ha terminado.

3.5. IPtables

IPtables es una poderosa y flexible herramienta de administración de cortafuegos (firewall) en sistemas operativos basados en Linux. Esta utilidad es fundamental para garantizar la seguridad y la gestión del tráfico de red en entornos Linux, proporcionando un control detallado sobre el manejo de los paquetes de datos que entran y salen del sistema.

IPtables opera utilizando una serie de tablas, cada una de las cuales contiene un conjunto de reglas específicas. Estas reglas determinan cómo se deben filtrar los paquetes, cómo se realiza la traducción de direcciones de red (NAT), y cómo se llevan a cabo otras operaciones de red cruciales. En esencia, IPtables permite definir políticas que controlan qué tráfico es permitido o denegado, contribuyendo significativamente a la protección contra accesos no autorizados y ataques potenciales.

Las tablas más comúnmente utilizadas en IPtables incluyen la tabla de filtro, que es la encargada de la filtración de paquetes, y la tabla de NAT, que maneja la traducción de direcciones de red. Además, hay otras tablas especializadas como la tabla mangle, que se utiliza para modificar los paquetes en tránsito, y la tabla raw, que permite configurar reglas antes de que el kernel inicie el seguimiento de conexiones.

Cada tabla en IPtables contiene varias cadenas (chains) predefinidas, como INPUT, OUTPUT y FORWARD, que representan diferentes puntos en el trayecto de los paquetes a través del sistema. Las reglas dentro de estas cadenas

especifican las acciones a tomar para cada paquete, como aceptarlo, rechazarlo, o descartarlo.

Una de las ventajas más destacadas de IPtables es su capacidad de configuración granular y detallada. Los administradores de sistemas pueden definir reglas basadas en una amplia variedad de criterios, incluyendo la dirección IP de origen o destino, el puerto de origen o destino, el protocolo utilizado, y muchos otros parámetros. Esto permite crear configuraciones de seguridad muy específicas y adaptadas a las necesidades particulares de cada red.

En resumen, IPtables es una herramienta esencial en la administración de cortafuegos en sistemas Linux, proporcionando una interfaz robusta y versátil para gestionar el tráfico de red y proteger el sistema contra amenazas. Su capacidad para manejar una amplia gama de operaciones de red y su alto grado de configurabilidad la convierten en una solución preferida para la seguridad y administración de redes en entornos Linux.

4 Desarrollo

Una vez explicado el contexto y los objetivos de este proyecto empezamos el desarrollo siguiendo la metodología anteriormente presentada.

4.1. Especificación de requisitos

En primer lugar vamos a definir los requisitos de nuestro servidor de forma específica.

1. El servidor debe alojar su propia red local.
2. La red local del servidor debe soportar la conexión simultanea de al menos 3 dispositivos.
3. El servidor debe conocer en todo momento la dirección IP asignada a cada dispositivo conectado a la red local y tener esta información disponible para la aplicación.
4. La red local debe permitir el intercambio bidireccional de audio en todos sus canales.
5. La red local debe tener un canal broadcast donde se permita la comunicación todos con todos de los dispositivos conectados.
6. La red local debe poder crear canales de comunicación 1 a 1 entre dos dispositivos conectados.
7. Los dispositivos conectados deben poder estar a su vez en un canal 1 a 1 y en el canal broadcast.
8. El servidor debe poder conectarse a un monitor para mostrar la aplicación.
9. El servidor debe permitir la conexión de un mouse para interactuar con la aplicación.
10. La aplicación debe mostrar en todo momento los dispositivos conectados a la red local.
11. La aplicación debe permitir silenciar dispositivos, conectados a la red local, del canal broadcast.
12. La aplicación debe permitir ensordecer dispositivos, conectados a la red local, del canal broadcast.
13. La aplicación debe poder añadir dispositivos, conectados a la red local, a canales de comunicación 1 a 1.
14. La aplicación debe permitir ensordecer o silenciar los dispositivos en un canal de comunicación 1 a 1.

4.2. Diseño de la interfaz de Usuario

Una vez conocemos los requisitos de nuestro sistema podemos empezar el diseño de nuestra aplicación. La aplicación estará desarrollada en Python usando la librería **TKinter** para diseñar la interfaz gráfica. En primer lugar desarrollaremos una versión preliminar de la interfaz únicamente para tener una idea clara de las necesidades y funcionalidades de nuestra aplicación. Luego, una vez completado el desarrollo del back-end de la aplicación se actualizará la interfaz para implementar todas las funcionalidades desarrolladas y hacerla mas amigable al usuario.

4.2.1. Versión preliminar de la interfaz

Ahora desarrollamos una versión muy básica de nuestra interfaz simplemente para definir la estructura de nuestra aplicación y las necesidades de comunicación entre la aplicación y el sistema operativo.

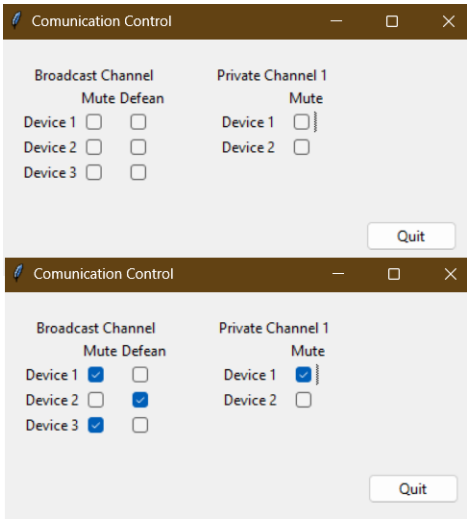

Fig. 3: Versión preliminar de la interfaz de usuario

Como podemos ver esta versión preliminar de la interfaz (Fig.3) la aplicación permite al usuario silenciar y ensordecer dispositivos del canal general y silenciar los dispositivos conectados a un canal privado. En esta versión esta lista de dispositivos es estática, de cara a la versión final esta lista mostrará en tiempo real todos los dispositivos conectados a la red local del servidor. El canal privado también es un ejemplo, en la versión final o bien tendrá un botón para añadir dispositivos o dispondrá de un *Drag & Drop* para mover dispositivos del canal general a uno privado.

4.2.2. Versión final de la interfaz

En esta versión de la interfaz (Fig. 4) contamos con las funciones necesarias para listar de forma dinámica los dispositivos conectados, los dispositivos conocidos se listaran con un nombre configurable mediante el archivo *devices.txt*. También siponemos de un botón de *refresh* para actualizar

esta lista. Finalmente hemos decidio prescindir del canal privado, en su lugar se dispone de otra pestaña con otras opciones de control, aquí podemos seleccionar 1 a 1 de forma individual con que walkies puede hablar cada dispositivo. En Fig. 5 vemos como el Walkie-1 podría comunicarse

Fig. 4: Versión final de la interfaz de usuario Broadcast Mode

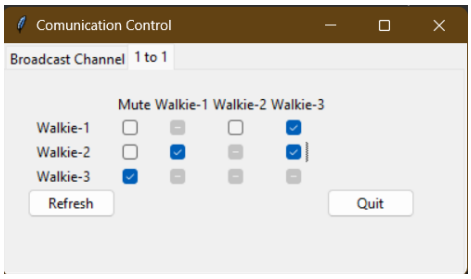

Fig. 5: Versión final de la interfaz de usuario 1 to 1 Mode

con el Walkie-3 pero no con el 2, el Walkie-2 puede comunicarse con normalidad con los otros dos dispositivos y el Walkie-3 esta muteado. Así pues el usuario del Walkie-3 escucharía a todos sus compañeros pero no podría hablarle a ninguno, el usuario del Walkie-2 no escucharía a nadie pero podría hablarle a los dos y el usuario del Walkie-1 escucharía únicamente al usuario del Walkie-2 y podría hablarle únicamente al del 3.

4.3. Configuración del host

Ahora vamos a configurar la Raspberry Pi Zero 2W. Deberemos instalarle un sistema operativo, configurar el host de la red local y preparar el enterno para que pueda ejecutar nuestra aplicación de control.

4.3.1. Instalación del Sistema Operativo en la Raspberry

Siguiendo la documentación de Raspberry [3] instalaremos el sistema operativo usando el software *Raspberry Pi Imager*. Necesitaremos una tarjeta SD de como mínimo 16GB, conectamos esta tarjeta al ordenador donde tengamos el *PI Imager* y arrancamos el software. Veremos la ventana que se muestra a la izquierda en Fig. 6. En la opción **Elegir dispositivo** seleccionamos **Raspberry Pi Zero 2W**, en **Elegir SO** elegiremos **Raspberry Pi OS (Legacy, 64-bit)** y finalmente en **Almacenamiento** seleccionamos la tarjeta SD en la que queremos instalar el sistema operativo. Al hacer clic en **Siguiente** nos aparecerá una ventana en la que haremos clic en **Editar ajustes** y configuraremos la sección **General** como se ve en la parte derecha de la Fig.

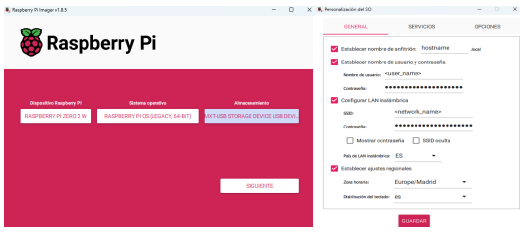

Fig. 6: Proceso de Desarrollo del Sistema de Comunicacion Local

6. Por último hacemos clic en **Guardar**, **Sí** y otra vez **Sí**. Ahora arrancamos nuestra Raspberry por primera vez, para ello le introducimos la tarjeta SD con el sistema operativo, lo conectamos a un monitor usando un cable micro-HDMI, usando un switch de USBs podremos conectar un mouse y un teclado al único puerto micro-USB para periféricos que tiene nuestra Raspberry, el otro puerto micro-USB ces el de alimentación, lo conectamos y vemremos como arranca el dispositivo con normalidad. (Fig. 7)

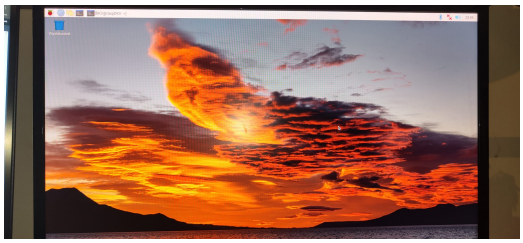

Fig. 7: Escritorio Rapberry Pi Zero 2W

4.3.2. Configuración de la Red Local

Para configurar la red local utilizaremos NetworkManager [5], debemos asegurarnos que este activo en nuestro sistema y configurar el punto de acceso como vemos en Fig. 8 Con esto ya deberíamos tener activa nuestra red local.

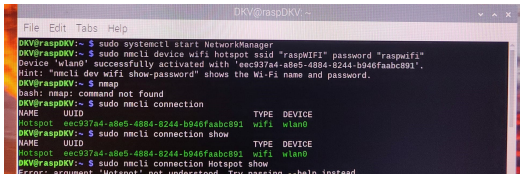

Fig. 8: Inicializando la red local

4.3.3. Verificación

Para verificar el correcto funcionamiento de la red podemos probar a conectarnos desde un ordenador o móvil, como vemos en (Fig. 9) la red esta visible i es posible conectarse a ella. Podemos apreciar también que el portátil indica que no tenemos conexión a Internet, esto es debido a que la Raspberry solo dispone de una interfaz la cual esta siendo utilizada para alojar la red local, así que no puede conectarse a otra red con acceso a Internet para hacer el reenvío de los paquetes. Otra forma de comprobar que los dispositivos se hayan conectado correctamente a la red local es consultar el archivo *"/proc/net/arp"* (Fig. 10). En este archivo se

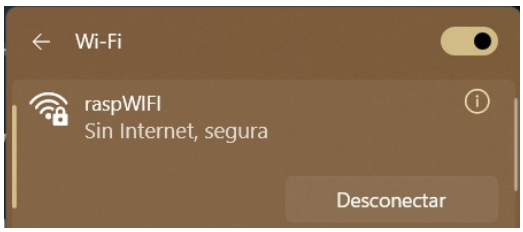

Fig. 9: Ordenador personal conectado a la red local.

pueden ver las direcciones MAC e IP de los dispositivos conectados, en este caso teníamos conectado el portátil y un móvil. También nos serviremos de este archivo para el desarrollo de la aplicacción. También podemos probar a hacer

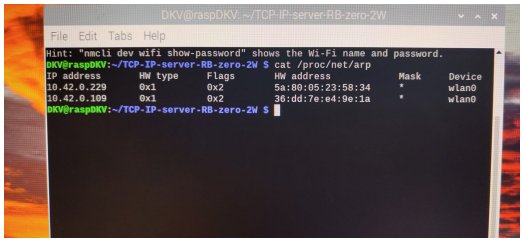

Fig. 10: Archivo ARP de la Raspberry

ping desde el PC al movil como vemos en (Fig. 11) para verificar que ambos dispositivos están conectados a la misma red.

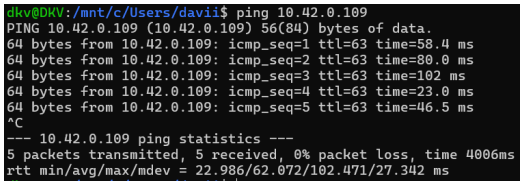

Fig. 11: Verificación mediante *Ping*

4.4. Rule-set de IPtables

Como hemos mencionado anteriormente para controlar las comunicaciones en nuestra red usaremos IPtables asi que en primer lugar comprobaremos el funcionamiento del módulo y luego desarrollaremos un rule-set que satisfaga todas las necesidades de nuestra aplicación.

4.4.1. Prueba de funcionamiento

En primer lugar probaremos a hacer *ping* pero esta vez a la raspberry en la cual hemos añadido una regla que bloquea todo el trafico entrante, como se ve en (Fig. 12) no recibimos respuesta. Ahora definimos una regla que reenvía

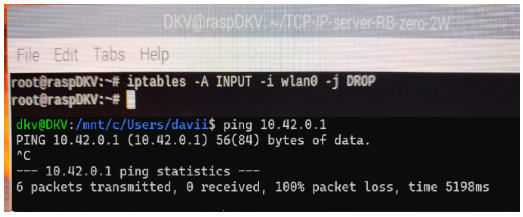

Fig. 12: Verificación DROP IPtables

el trafico TCP a la dirección IP del móvil. Enviando un paquete TCP usando netcat desde el ordenador dirigido a la raspberry y escuchando, tambien con netcat, en el movil podemos ver como es este el que recibe y responde a los paquetes. (Fig. 13)

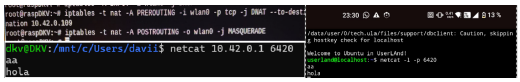

Fig. 13: Verificación reenvío datos

4.4.2. Implementación

Necesitaremos un conjunto de reglas que nos permita silenciar a los dispositivos de forma individual, para eso bloquearemos el trafico proveniente de una dirección IP concreta, la cual tendremos disponible en la aplicación gracias a la lista de dispositivos dinámica, y con destino al puerto 6420 que es al cual los walkie-talkies envían el audio (Fig. 14). El motivo de bloquear el trafico solo con destino al puerto 6420 es que los walkies usan también el puerto 6421 para recibir audio, a través de este puerto deben llegar los mensajes ACK para el correcto funcionamiento del protocolo TCP/IP, por tanto no podemos bloquear todo el trafico de una IP porque no solo lo silenciaríamos si no que también impediríamos que recibiese audio.

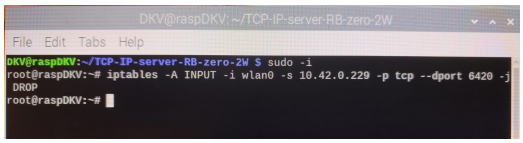

Fig. 14: Ejemplo regla DROP para silenciar un dispositivo

4.4.3. Verificación

Para verificar el correcto funcionamiento de estas reglas en IPtables, podemos definir una regla específica que silencie un dispositivo determinado. Luego, realizaremos varias pruebas para asegurar que la regla se está aplicando correctamente.

Primero, configuramos una regla en IPtables que silencie el dispositivo en cuestión, bloqueando todo el tráfico hacia. Una vez establecida esta regla, la primera prueba consiste en intentar enviar un paquete desde el dispositivo silenciado al puerto 6420. Si la regla está funcionando correctamente, este paquete debería ser bloqueado y no debería llegar a su destino.

A continuación, realizamos una segunda prueba para confirmar que el dispositivo silenciado puede enviar paquetes a otros puertos. Por ejemplo, intentamos hacer un ping desde este dispositivo a otro sistema. Este test nos ayudará a verificar si la regla está afectando solamente al puerto 6420 o si está interfiriendo con otras comunicaciones del dispositivo.

Finalmente, para completar la verificación, enviamos un paquete al puerto 6420 desde un dispositivo diferente, que no esté silenciado. Esta prueba es crucial para asegurar que el puerto 6420 sigue siendo accesible desde otros dispositivos de la red y que la regla se aplica únicamente al dispositivo especificado inicialmente.

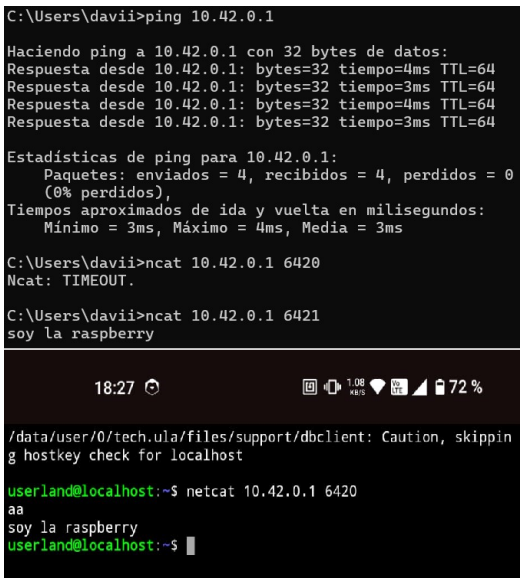

Fig. 15: Verificación silenciar dispositivos

Al realizar estas tres pruebas—intentar enviar un paquete al puerto 6420 desde el dispositivo silenciado, enviar paquetes desde el dispositivo silenciado a otros puertos, y enviar un paquete al puerto 6420 desde un dispositivo no silenciado—podemos confirmar de manera exhaustiva el correcto funcionamiento de las reglas definidas en IPtables. Esta metodología no solo valida que las reglas se aplican según lo previsto, sino que también garantiza que el resto de la red continúa operando normalmente sin interferencias no deseadas.

4.5. Desarrollo aplicación

Con la versión preliminar de la interfaz y las necesidades de control junto con el rule-set de IPtables definidos podemos desarrollar el back-end de nuestra aplicación.

4.5.1. Setup de la configuración de red

Lo primero que deberá hacer nuestra aplicación al lanzarse es configurar la red local de la Raspberry tal como hemos visto anteriormente, para ello usaremos la librería Os que nos permite hacer llamadas al sistema operativo, la función de inicializacion es la siguiente:

```
16 def setupNetwork():
17     os.system("sudo systemctl start NetworkManager")
18     time.sleep(5)
19     os.system("sudo nmcli device wifi hotspot ssid 'raspWIFI' password 'raspwifi'")
```

Fig. 16: Función para inicializar la red local

4.5.2. Listado de dispositivos

La aplicación debe mostrar al usuario la lista de dispositivos conectados de una forma que pueda reconocer cual es cual y a su vez asociar el dispositivo de la lista con su dirección IP para la gestión de las IPtables. Para ello usaremos el archivo *"/proc/net/arp"* que hemos visto anteriormente en el cual vemos las direcciones IP asociadas a las direcciones MAC, esto nos sirve porque la dirección IP puede cambiar cada vez que el dispositivo se conecta a la red pero la dirección MAC es estática y propia de la tarjeta de conectividad del dispositivo. Crearemos otro archivo (*devices.txt*) en el que guardaremos una lista de las direcciones MAC asociadas al nombre del dispositivo, de esta forma cuando se inicie la aplicación consultará el listado de dispositivos conectados en el archivo *"/proc/net/arp"* para obtener las direcciones IP, buscará las direcciones MAC en el archivo *"devices.txt"* y así tendremos las IP asociadas al nombre para mostrar la lista al usuario y poder gestionar las IPtables para cada dispositivo. Si en *"/proc/net/arp"* vemos que hay un dispositivo conectado el cual no esta en *"devices.txt"* este se mostrara en la aplicación como *"Unknown"*.

4.5.3. Integración IPtables para silenciar

Para silenciar dispositivos, usaremos los *mute checkbuttons* de nuestra interfaz, los cuales cuando se activan añaden la regla de DROP específico para la IP del dispositivo al cual haga referencia ese *chackbutton*, así pues si el botón esta activo, la regla lo estará y el dispositivo estará mutado ya que todos los paquetes que envíe al puerto 6420 serán descartados, cuando el botón esta inactivo la regla se elimina y los paquetes vuelven a llegar a su destino con normalidad.

4.5.4. Reenvío de paquetes mediante sockets y ensordecimiento de dispositivos

Para recibir los paquetes TCP enviados al puerto 6420 de la Raspberry crearemos un socket que escuche en este puerto, las conexiones entrantes se trataran con la función socket.accept() la cual devuelve otro socket el cual es el punto final de la comunicación con el cliente y el primer socket permanece a la escucha en el mismo puerto para aceptar otras conexiones. También cada vez que se acepte una conexión se creara un socket, además del que crea la función socket.accept(), este socket lo conectaremos al puerto 6421 del walkie que nos haya iniciado la conexión, utilizaremos este socket para enviarle a este walkie el audio proveniente de los demás dispositivos. Así pues todo lo que nos llegue por un socket servidor (puerto 6420) lo reenviaremos por todos los socket cliente que tenemos menos el del dispositivo origen del mensaje. Cuando queramos ensordecer un dispositivo, en el caso del modo broadcast activando el *checkbutton* lo que haremos sera quitar el socket conectado al puerto 6421 de este dispositivo del array de sockets al cual se reenvían los paquetes, el modo de comunicación individual es un poco mas complejo ya que solo deberemos dejar de reenviar lo que proviene de un dispositivo en concreto, en este caso se contara con un vector para cada dispositivo conectado en el cual cada posición representa uno de los otros dispositivos y el valor en esa posición (0 o 1) determinara si se le reenvía o no los paquetes a ese dispositivo.

5 Resultados

Una vez finalizado el desarrollo tanto del servidor como de los walkie-talkies (trabajo desarrollado en el Trabajo de Fin de Grado de Informática), procederemos a verificar el correcto funcionamiento del sistema en su totalidad. Para ello, necesitaremos contar con tres walkie-talkies y una Raspberry Pi. El primer paso será conectar la Raspberry Pi

y lanzar la aplicación del servidor. Al hacerlo, se establecerá la red Wi-Fi, dejándola operativa y lista para su uso.

Una vez que la red Wi-Fi esté activa, encenderemos los walkie-talkies y verificaremos que se conecten automáticamente a la red. En esta fase, es crucial asegurarse de que la conexión se establezca de forma fluida y sin intervención manual adicional. Después de la conexión, procederemos a comprobar la comunicación entre los walkie-talkies. Observamos que los dispositivos se comunican correctamente entre sí. Aunque la calidad del audio no es óptima debido a la limitación del DAC de 8 bits del ESP32, resulta perfectamente comprensible y permite una comunicación efectiva entre los usuarios.

El alcance de los walkie-talkies está condicionado por el rango de la red Wi-Fi proporcionada por la Raspberry Pi, que es de aproximadamente 25 metros. Esto significa que dos usuarios pueden comunicarse a una distancia máxima de 50 metros. Este rango de comunicación es adecuado para el uso previsto en salas de escape (escape rooms), que fue el contexto inicial para el cual se diseñó este sistema.

Además, el control de la comunicación se gestiona de manera completa desde la aplicación del servidor. Podemos realizar diversas pruebas para asegurarnos de que todas las funcionalidades se comportan según lo esperado. Por ejemplo, podemos silenciar un dispositivo específico y verificar que los otros dos puedan seguir comunicándose sin que el dispositivo silenciado escuche nada. También podemos crear un entorno en el que los usuarios se comuniquen a través de los walkie-talkies con una dinámica similar a la del juego del ”teléfono escacharrado”. Esta funcionalidad puede ser especialmente útil para actividades dentro de las salas de escape, añadiendo un elemento de diversión y desafío a la experiencia del usuario.

En conclusión, podemos afirmar que nuestro sistema cumple con las expectativas previstas. La implementación y las pruebas realizadas demuestran que el sistema es funcional y adecuado para su uso en las condiciones y contextos planteados. Este proyecto no solo ha cumplido con los objetivos iniciales, sino que también ha proporcionado una herramienta efectiva y entretenida para los participantes en las actividades de las salas de escape, asegurando una comunicación fluida y controlada.

6 Conclusiones

Las conclusiones de este trabajo reflejan un proceso de aprendizaje significativo y el cumplimiento exitoso de los objetivos planteados. A lo largo del desarrollo del sistema de comunicación local mediante walkie-talkies TCP/IP, se han adquirido conocimientos valiosos en diversas áreas de la ingeniería informática y las telecomunicaciones.

En primer lugar, el proyecto ha demostrado que es posible diseñar e implementar un sistema de comunicación eficiente y controlado utilizando tecnologías digitales modernas. El uso de una Raspberry Pi Zero 2W como servidor ha sido una decisión acertada, ya que esta pequeña pero potente herramienta ha permitido alojar la red local y gestionar las comunicaciones de manera efectiva. Además, la integración de IPtables para controlar el tráfico de red ha mostrado ser una solución robusta y eficiente, especialmente en un dispositivo con recursos limitados como la Raspberry Pi.

El desarrollo de la interfaz gráfica en Python, utilizando Tkinter, ha sido otro aspecto destacado del proyecto. Esta experiencia ha permitido profundizar en el diseño de interfaces de usuario amigables, facilitando la gestión de las comunicaciones sin necesidad de conocimientos avanzados en administración de redes. La creación de una aplicación que interactúa directamente con el sistema operativo para modificar las reglas de IPtables ha demostrado la importancia de la interacción entre software y hardware en la implementación de sistemas complejos.

Durante el proceso, se ha aprendido a valorar la importancia de las pruebas exhaustivas. Las diversas pruebas de funcionalidad realizadas han confirmado que el sistema se comporta según lo esperado, destacando la capacidad de silenciar y ensordecer dispositivos específicos, así como de gestionar comunicaciones independientes. Este enfoque metodológico ha subrayado la necesidad de validar cada componente del sistema para asegurar su correcto funcionamiento en conjunto.

El contexto educativo de este proyecto también ha resaltado la aplicabilidad de los conocimientos teóricos en situaciones prácticas. El diseño y desarrollo del sistema para su uso en salas de escape ha proporcionado un entorno realista y desafiante, permitiendo aplicar conceptos de redes y programación en un escenario concreto. Este enfoque práctico ha facilitado una comprensión más profunda de los principios subyacentes y ha mostrado cómo las soluciones tecnológicas pueden mejorar experiencias interactivas en la vida real.

En conclusión, este trabajo no solo ha logrado cumplir con los objetivos técnicos propuestos, sino que también ha enriquecido el conocimiento y las habilidades en ingeniería informática. La experiencia adquirida en el desarrollo de un sistema de comunicación controlado y eficiente, utilizando herramientas modernas y un enfoque práctico, ha sido invaluable desde una perspectiva educativa. Este proyecto sirve como un ejemplo claro de cómo la teoría y la práctica pueden integrarse para crear soluciones innovadoras y efectivas en el campo de las telecomunicaciones.

Referencias

[1] Raspberry Pi Ltd, “Raspberry Pi Documentation” `https://www.raspberrypi.com/documentat*ion/computers/os.html#introduction` 2024

[2] Raspberry Pi Foundation. [Raspberry Pi Zero 2 W] `https://www.raspberrypi.org/products/raspberry-pi-zero-2-w/`

[3] Raspberry Pi Ltd, “Raspberry Pi Documentation” `https://www.raspberrypi.com/documentation/computers/getting-started.html#installing-the-operating-system` 2024

[4] Python Software Foundation, “Interfaces gráficas de usuario con Tk” `https://docs.python.org/es/3/library/tk.html` 2024

[5] NetworkManager
`https://networkmanager.dev/` 2024

[6] Tanenbaum, A. S., & Wetherall, D. J. (2011). Computer Networks (5thed.). Prentice Hall.

APÉNDICE

A.1. Code

Todo el codigo tanto del servidor como el firmware del walkie esta disponible en `https://github.com/Destroy65/TCP-IP-server-RB-zero-2W`

A.2. Especificaciones técnicas Raspberry Pi Zero 2W

La Raspberry Pi Zero 2W es una mejora significativa respecto a su predecesora, la Raspberry Pi Zero W. Sus especificaciones técnicas clave son:

- **Procesador:** Broadcom BCM2710A1, quad-core Cortex-A53 a 1 GHz
- **Memoria:** 512 MB de SDRAM LPDDR2
- **Conectividad inalámbrica:**
  - Wi-Fi 802.11 b/g/n (2.4 GHz)
  - Bluetooth 4.2, BLE
- **Puertos y conectividad:**
  - 1 puerto micro-USB OTG
  - 1 puerto micro-USB para alimentación
  - Mini HDMI
  - Conector CSI-2 para cámara
  - Ranura microSD para almacenamiento
- **Consumo de energía:** Aproximadamente 0.5 W en reposo