Taller 3: ROS (Robot Operating System) sensor de temperatura (Serial)

3.1 Objetivos

- Comprender como ROS interactúa con sensores externos.
- Identificar el funcionamiento de un equipo Arduino junto con ROS.
- Realizar la conexión entre Arduino y nodos de ROS.

3.2 Marco conceptual

3.2.1 Arduino

Arduino es una plataforma de código abierto (Open-Source) que se ha convertido en un estándar global para el desarrollo de proyectos de electrónica interactiva. Su filosofía se basa en la combinación de dos elementos fundamentales que facilitan la creación de prototipos y sistemas de control: Hardware y Software libres y flexibles. El hardware físico de Arduino se materializa en una placa de circuito impreso que incorpora un microcontrolador (e.g., ATmegaXXX) programable del sistema. El software, por su parte, consiste en el Entorno de Desarrollo Integrado (IDE), que es una aplicación multiplataforma que permite a los usuarios escribir código (basado en el lenguaje de programación Wiring, similar a C++) y cargar esas instrucciones directamente en el microcontrolador de la placa. Esta dualidad de acceso libre al diseño de hardware y al código fuente ha impulsado una vasta comunidad de usuarios y una gran versatilidad de aplicaciones.

3.2.2 Principios de funcionamiento:

El Arduino se centra en el ciclo de interacción con el entorno físico. La placa, una vez programada con un sketch (código), opera de manera autónoma, sin necesidad de permanecer conectada al ordenador.

1. **Entrada de Datos:** El microcontrolador recibe información del mundo exterior a través de sus pines de entrada/salida (I/O). Estos pines se conectan típicamente a sensores (dispositivos que miden variables físicas como luz, temperatura, distancia, etc.).
2. **Procesamiento:** El código cargado en la memoria del microcontrolador (generalmente un ATmega) ejecuta una serie de instrucciones lógicas. El microcontrolador evalúa las señales de entrada recibidas y toma decisiones basadas en la programación definida.
3. **Salida de Acción:** Como resultado del procesamiento, el microcontrolador genera señales de salida a través de sus pines I/O, las cuales son enviadas a actuadores (dispositivos que realizan una acción física, como encender un LED, activar un motor, mostrar información en una pantalla, etc.).

Este ciclo continuo de sensar, procesar y actuar permite crear sistemas de control y automatización que interactúan directamente con el ambiente.

3.2.3 Equipos Arduino:

Arduino no es un único modelo de placa, sino una plataforma con diversas variantes que se adaptan a distintas necesidades de proyecto, diferenciándose principalmente en su microcontrolador, tamaño y número de pines. Algunos de los tipos más representativos incluyen:

1. **Arduino Uno:** El modelo más popular y la placa de referencia para principiantes. Equilibrio entre simplicidad, rendimiento y costo.
2. **Arduino Mega:** Orientado a proyectos complejos que requieren una gran cantidad de conexiones. Posee un microcontrolador más potente (ATmega2560) con un mayor número de pines digitales y analógicos. Se utiliza frecuentemente en proyectos de robótica avanzada o en impresoras 3D.
3. **Arduino Leonardo:** Una variante que utiliza un microcontrolador ATmega32U4, lo que le permite emular directamente un dispositivo USB (como un teclado o un ratón), simplificando la interacción con ordenadores.

En el presente proyecto, usaremos el Arduino UNO; por lo que nos enfocaremos en él. El Arduino Uno es el estandarte de la plataforma, elegido por su robustez, facilidad de uso y la vasta documentación disponible (ver Tabla 3.1).

Características Técnicas	**Especificaciones**
`Microcontrolador`	ATmega328P
`Voltaje de Funcionamiento`	5V
`Pines Digitales I/O`	14 (6 de los cuales pueden usarse como salidas PWM)
`Pines de Entrada Analógica`	6 (Etiquetados como A0 a A5)
`Velocidad de Reloj`	16MHz

Table 3.1: Características de Arduino UNO

Las Entradas Analógicas del Arduino Uno, pines A0 a A5, son cruciales para la interacción con el mundo real, ya que permiten la lectura de señales de voltaje continuo variables. El microcontrolador ATmega328P integra un Convertidor Analógico-Digital (ADC) de 10 bits de resolución. Este componente es el encargado de transformar una señal de voltaje analógica (continua) proveniente de un sensor en un valor digital discreto (binario) que el microcontrolador pueda procesar.

El proceso de conversión es el siguiente:

1. **Rango de Medición:** Por defecto, el ADC mide voltajes en un rango de 0V a 5V (el voltaje de operación de la placa).
2. **Resolución:** Un ADC de 10 bits significa que el rango de voltaje (5V) se divide en $2^{10} = 1024$ niveles o pasos discretos (de 0 a 1023).
3. **Cuantificación:** El valor digital leído por la función analogRead() se calcula mediante la siguiente relación:

$$\text{Valor Leido} = \left\lfloor \frac{\text{Voltaje Referencia}}{\text{Voltaje de entrada}} \times 1024 \right\rfloor$$

La placa Arduino UNO utiliza un convertidor USB a Serial, también conocido como UART, para manejar la comunicación a través del cable USB. El Arduino tiene un chip

dedicado cuya función es tomar la comunicación serial de nivel TTL (Transistor-Transistor Logic, 5 V o 3.3 V) y convertirla al protocolo USB para que la PC pueda entenderla. Cuando conecta el Arduino al PC, el sistema operativo (Windows, macOS, Linux) lo reconoce y le asigna un Puerto de Comunicación virtual, generalmente etiquetado como COMx (en Windows) o /dev/tty.usbmodemX (en macOS o Linux).

3.2.4 Comunicación serial en Arduino

La comunicación serial es uno de los métodos más fundamentales y útiles para interactuar con una placa Arduino desde un ordenador (PC). Permite enviar datos desde el Arduino a la PC (por ejemplo, lecturas de sensores) o enviar comandos desde la PC al Arduino (por ejemplo, para controlar un LED).El protocolo serial es asíncrono, lo que significa que el transmisor y el receptor no comparten una señal de reloj. En su lugar, ambos deben estar configurados a la misma velocidad de transmisión (baudios) para que los datos (bits) se interpreten correctamente. En el Arduino Uno, los pines del puerto serial D0 y D1 están internamente conectados al chip conversor USB a Serial, por lo que no se debe conectar nada externo a los pines mencionados anteriormente mientras se usa la comunicación USB con la PC, ya que interferiría con la carga del sketch (programa) y la comunicación con el Monitor Serial.(ver Fig.3.1)

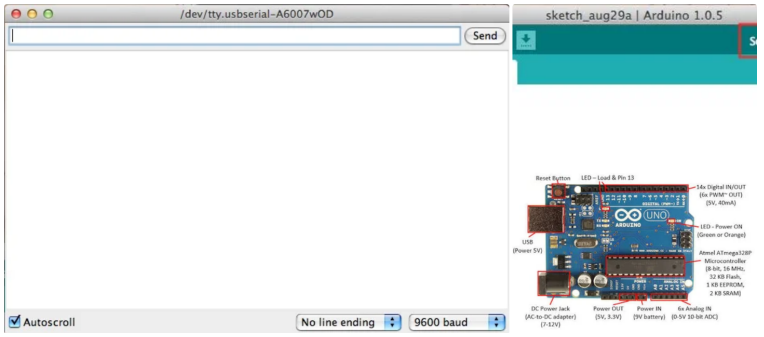

Figure 3.1: Conexión Arduino PC Serial IDE

La herramienta clave para interactuar con la comunicación serial en la PC es el Monitor Serial, integrado en el IDE de Arduino. Luego de que su sketch esté cargado en el Arduino y la placa esté conectada por USB, haga clic en el icono de la lupa en la esquina superior derecha del Arduino IDE, o vaya a Herramientas > Monitor Serial. La velocidad es de 9600 baudios, que es la más utilizada por defecto. El Arduino y el Monitor Serial del IDE deben usar la misma velocidad.

3.3 Comunicación en ROS

La comunicación en ROS se basa en DDS y RTPS. En la Tabla 3.2 ubicamos a DDS en capas superiores, mientras que RTPS se encarga del transporte.

- **DDS (Data Distribution Service)** `https://www.dds-foundation.org/what-is-dds-3/` es un middleware de comunicación distribuida estandarizado por la OMG (Object Management Group `https://www.omg.org/spec/DDSI-RTPS/2.2/About-DDSI-RTPS`).

Modelo TCP/IP	Ejemplo / Función	Relación con DDS / RTPS
Aplicación	Nodos ROS 2 (publisher, subscriber, etc.)	Usan **DDS** para manejar los datos, tópicos y QoS
Transporte	**UDP o TCP**	RTPS normalmente usa **UDP**
Red (Internet)	**IP** (direcciones y enrutamiento)	RTPS se apoya en **direcciones IP** para enviar paquetes
Enlace / Física	Ethernet, Wi-Fi, etc.	Transmiten los paquetes IP físicamente

Table 3.2: Relación entre las capas TCP/IP y los protocolos DDS y RTPS en ROS 2.

DDS sigue el modelo de publicación/suscripción (publish/subscribe). Todo ocurre sin un maestro central; es decir, los nodos se descubren y comunican de manera descentralizada.
  - Un nodo publicador (publisher) envía mensajes en un tópico.
  - Un nodo suscriptor (subscriber) recibe esos mensajes si está suscrito al mismo tópico.

  Además, DDS se encarga de:
  - QoS (Quality of Service): permite definir confiabilidad, prioridad, tiempo de vida del mensaje, etc.
  - Descubrimiento automático: los nodos se encuentran entre sí sin necesidad de configuración manual.
  - Entrega eficiente: se pueden usar multicast, unicast o ambos, según la red.
- **RTPS (Real-Time Publish-Subscribe Protocol)**: DDS usa RTPS como protocolo de transporte para comunicarse a través de la red. RTPS define cómo se empaquetan y envían los mensajes (nivel de red, ver Tabla 3.2). RTPS, por defecto, opera sobre UDP/IP. Usa mensajes binarios definidos por el estándar RTPS. Permite la comunicación en tiempo real y determinista, lo cual es crucial en la robótica.

3.3.1 Flujo de información

Considere los dos nodos utilizados en la práctica anterior. Primero, tenemos un sensor_node que publica en el tópico /temperature, y un segundo nodo reader_node se subscribe al tópico /temperature. La comunicación ocurre en dos pasos:

- **Paso 1**: Descubrimiento automático (RTPS)
  Cuando ambos nodos se ejecutan, Cada uno envía mensajes RTPS de anuncio (discovery) por multicast. Los demás nodos en la red los reciben. Si encuentran coincidencias (mismo tópico y tipo de mensaje), establecen un enlace lógico. DDS/RTPS se encarga del descubrimiento (Distributed).
- **Paso 2**: Intercambio de datos (DDS sobre RTPS)
  Una vez descubiertos: El publisher serializa los datos del mensaje ROS 2. DDS encapsula ese mensaje en un paquete RTPS. El paquete se envía por UDP al suscriptor (o por multicast si hay varios). El suscriptor deserializa el mensaje y lo entrega al nodo de ROS 2.

3.3.2 Ventajas de usar DDS + RTPS en ROS 2

- Comunicación descentralizada (sin maestro).
- Escalable: muchos nodos y máquinas pueden interactuar.
- Confiable y configurable mediante QoS.
- Compatibilidad en tiempo real (determinismo).
- Interoperabilidad entre distintos fabricantes (por ser un estándar abierto).

3.4 Desarrollo

3.4.1 ROS y DOCKER, COMUNICACIÓN SERIAL

Para esta sección, utilizaremos un equipo arduino para comunicarnos de manera serial con ROS. La idea es un nodo que interprete lo que me envía el equipo arduino de manera serial; el segundo nodo transforma dichos datos en valores de una temperatura simulada, y por último, el tercer nodo muestra la temperatura ya transformada. Aprenderemos funciones como activar el puerto serial dentro de un contenedor Docker y reforzaremos los conocimientos sobre la estructura del ROS y su compilador COLCON. Con esto, verificaremos la comunicación entre ROS-DOCKER y un sensor serial. (ver Fig.3.2)

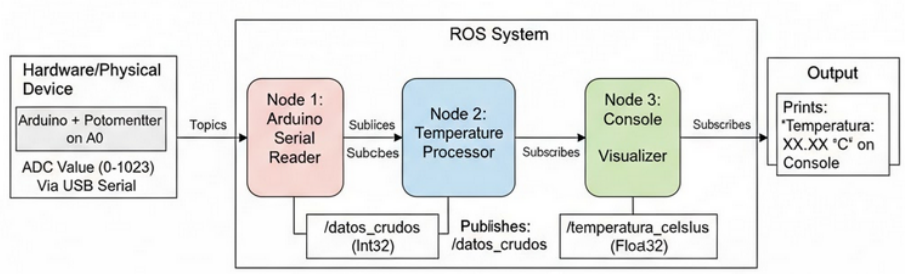

Figure 3.2: Esquema de la Practica

- **Preparar el Sensor**: Usaremos una entrada analógica del Arduino como ejemplo de simulación de un sensor de temperatura. En este caso, la entrada A0 recibirá la señal del sensor ejemplificado con un potenciómetro. El equipo transmitirá de forma serial el valor crudo a la computadora. En este caso, usaremos el `Arduino YUN`, muy similar al `Arduino UNO`. Desarrollaremos el código en Python para el IDE de Arduino.

```
// Arduino: Lee potenciometro en A0 y env a por Serial

int sensorPin = A0;
int sensorValue = 0;

void setup() {
  Serial.begin(9600);
}
void loop() {
  sensorValue = analogRead(sensorPin);
  Serial.println(sensorValue);
  delay(1000);  // cada segundo
}
```

Podemos usar el monitor serial del IDE de Arduino para verificar la información. Lo principal es identificar el puerto en el cual transmite el equipo Pueden presentarse opciones como `/dev/ttyACM0` ó `/dev/ttyUSB0` en Linux ; o el Hyperterminal o la herramienta `wmic` en Windows.

3.4.2 Preparar DOCKER para que detecte el puerto serial

Si utiliza Windows, para habilitar el acceso al puerto serial dentro del contenedor, abra una terminal (PowerShell) con privilegios de administrador e instale ***usbipd***, una herramienta oficial de Microsoft que permite compartir dispositivos USB con WSL2 (y, por consiguiente, con Docker Desktop).

Esta utilidad, denominada usbipd-win, posibilita redirigir dispositivos USB (como un Arduino) desde Windows hacia el entorno WSL2, de modo que el puerto serial aparezca en Linux como /dev/ttyACM0.

Sin esta configuración, Docker no podrá acceder al puerto serial del sistema.

Después de la instalación, cierre la terminal para guardar los cambio y lance otra. Ejecute ***uspipd list*** y debería ver algo similar a lo siguiente:

```
BUSID  DEVICE
1-5    Arduino Uno (COM3)
1-6    USB Keyboard
```

Una vez que tu Arduino aparece en la lista (1-5 es solo un ejemplo), procedemos a adjuntar el BUSID correspondiente al Docker con el comando *sbipd attach –wsl –busid 1-5*.
Los comandos son los siguientes:

```
winget install usbipd
usbipd list
usbipd attach --wsl --busid 1-5
```

Si está usando Linux como host nativo de los Dockers o una vez que ya ejecutó el codigo anterior en Windows, en una terminal de WSL, bastará con identificar el puerto serie que utiliza el Arduino. Ejecute las siguientes tareas según sea su caso:

1. Identifique el puerto serie que utilizará Arduino en WSL:

   ```
   dmesg | grep tty
   /dev/ttyACM0
   ```

2. (**Opcional**) Si usa un Linux como host, o un usuario diferente al root, debe otorgar permisos al puerto serie. En Linux, los puertos serie pertenecen al grupo `dialout`. Ejecutar la siguiente instrucción y reiniciar la sesión de Linux.

   ```
   sudo usermod -a -G dialout $USER
   ```

3. Por último puede ejecutar el Docker con la opción `-device` que permita conectarse directamente al puerto serie.

```
docker run -it --name ros2_sniff --device=/dev/ttyACM0 osrf/ros:
    jazzy-desktop bash
```

Dentro del contenedor instale el controlador serial en el contenedor.

```
apt update && apt install -y python3-serial
```

4. Verificar que Docker tenga comunicación con el puerto serial. Se pueden usar las herramientas `minicom` o `screen` dentro del Docker para verificar que recibe la información.

```
apt update && apt install -y minicom
minicom -D /dev/ttyACM0 -b 9600
```

3.5 Workspace y nodos

Al igual que en el taller anterior, el siguiente paso es crear nuestro espacio de trabajo y los paquetes necesarios para nuestros nodos.

```
cd ~/ros2_ws/src
ros2 pkg create --build-type ament_python sensor_serial
```

Eso crea la estructura básica:

```
sensor_serial/
           package.xml
           setup.py
           sensor_serial/
               __init__.py
               sensor_node.py              //vamos a crear
               processor_node.py           //vamos a crear
               monitor_node.py             //vamos a crear
```

- **Nodo sensor - Lectura de datos**: Lo primero que se desarrollará será el nodo que lea la información del puerto serial y la guarde en un tópico para su posterior uso.

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Int32
import serial

class SensorNode(Node):
    def __init__(self):
        super().__init__('sensor_node')
        self.ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)  //
            Puerto de Arduino
        self.publisher_ = self.create_publisher(Int32, 'sensor_data',
            10) //Topic - sensor_data
        self.timer = self.create_timer(1.0, self.publish_data)

    def publish_data(self):
        line = self.ser.readline().decode('utf-8').strip()
        if line.isdigit():  # asegura que es solo n meros
            value = int(line)
            msg = Int32()
```

```
            msg.data = value
            self.publisher_.publish(msg)
            self.get_logger().info(f' Publicando crudo: {value}')
        else:
            self.get_logger().warn(f' Dato inv lido: "{line}"')

def main(args=None):
    rclpy.init(args=args)
    node = SensorNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

- **Nodo Procesador - Transformación de magnitudes**: El nodo lee el tópico `sensor-data` y lo transforma en valores de temperatura. Almacena los valores en el tópico `temperature_celsius`.

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Int32, Float32

class ProcessorNode(Node):
    def __init__(self):
        super().__init__('processor_node')
        self.subscription = self.create_subscription(Int32, '
            sensor_data', self.listener_callback, 10)
        self.publisher_ = self.create_publisher(Float32, '
            temperature_celsius', 10)

    def listener_callback(self, msg):
        raw_value = msg.data
        temperature = (raw_value / 1023.0) * 100.0  # escala a 0
            100C
        temp_msg = Float32()
        temp_msg.data = temperature
        self.publisher_.publish(temp_msg)
        self.get_logger().info(f'Procesado: {temperature:.2f}  C ')

def main(args=None):
    rclpy.init(args=args)
    node = ProcessorNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

- **Nodo Monitor - Mostrar Información**: Muestra en consola la temperatura. Como en el ejemplo anterior, cada nodo corre en una ventana de terminal, por lo que usaremos `docker exec`.

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import Float32
```

```
class MonitorNode(Node):
    def __init__(self):
        super().__init__('monitor_node')
        self.subscription = self.create_subscription(Float32, '
            temperature_celsius',
            self.listener_callback, 10)
    def listener_callback(self, msg):
        self.get_logger().info(f' Temperatura actual: {msg.data:.2f}
            C ')

def main(args=None):
    rclpy.init(args=args)
    node = MonitorNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

- **Editar el archivo de configuración de Setup de ROS**: Agregar los puntos de entrada en el archivo de configuración `Setup.py`.

```
entry_points={
    'console_scripts': [
        'sensor_node = sensor_serial.sensor_node:main',
        'processor_node = sensor_serial.processor_node:main',
        'monitor_node = sensor_serial.reader_node:main',
    ],
}
```

- **Compilar y ejecutar**: Una vez desarrollados los nodos, procedemos a compilar y ejecutar el espacio de trabajo.

```
cd ~/ros2_ws
colcon build --packages-select sensor_serial

source /opt/ros/jazzy/setup.bash      // En cada ventana de terminal
source /install/setup.bash             // En cada ventana de terminal

 // Ejecutar cada nodo en un terminal diferente. Recordar que si
     usamos el opcion Docker exec , en cada contenedor debemos definir
      el source del proyecto y de ROS2

ros2 run sensor_reader sensor_node
ros2 run sensor_reader processor_node
ros2 run sensor_reader monitor_node
```

Con esto, hemos diseñado un sensor en Arduino (A0) que, a través del medio Serial-USB , transmite datos a ROS2 en el nodo sensor, que los interpreta y los envía al nodo procesador, donde se transforman en temperatura y, finalmente, se muestran en el nodo monitor en consola.

3.6 Analizando trafico con WIRESHARK

Wireshark es un analizador de paquetes de red gratuito y de código abierto que captura y muestra datos de forma detallada a medida que pasan por una conexión de red. Esta herramienta nos permite también analizar el tipo de paquetes de comunicación que se da entre nodos ROS, puerto serial, contenedores en una misma o diferentes redes;es decir, permite analizar mas a profundidad los mensajes de comunicación de los diferentes elementos de nuestra red.

Desarrollaremos un ejemplo de comunicación simple para familiarizarnos con Wireshark y su análisis en ROS. El escenario plantea el ejemplo del taller anterior, donde teníamos:

- **Nodo Sensor:** Simula un sensor real y genera temperaturas aleatorias
- **Nodo Receptor:** Simula un nodo que lee las temperaturas y las muestra en pantalla.

Para el análisis de tráfico, adicionaremos un tercer nodo que sera `Nodo-sniff`, que monitorizara todo el trafico entre los dos nodos.

Para aprovechar la interfaz gráfica de Wireshark, puede usar Windows como sistema operativo y ejecutar ROS dentro de Docker. De esta forma, el nodo "sniff" generará un archivo ***.pcap***, que luego podrá abrir en Wireshark y analizar cómodamente.

Cuando usa Docker en Windows, cada contenedor no corre "directamente" en Windows, sino dentro de una VM Linux interna (Docker Desktop la gestiona usando WSL2). Dentro de esa mini máquina Linux, cada contenedor ve una interfaz de red llamada eth0; esta no es la tarjeta física de su PC, sino una interfaz virtual conectada a un bridge interno de Docker.

Dentro de un contenedor, eth0 representa su conexión al bridge de la red Docker donde están sus nodos ROS2. Por lo tanto, dentro del contenedor sniff, se puede ver el tráfico entre los otros contenedores, siempre que estén en la misma red Docker bridge.

3.6.1 Pasos para analizar el trafico :

1. **Crear una red personalizada de Docker.** Crearemos una red llamada "***proyecto***".
2. **Agregar los contenedores de nuestro taller a la red creada en el punto anterior.**
3. **Ejecutar el contenedor sniff dentro de la misma red de docker**. Ejecutamos el contenedor con la imagen de nicolaka/netshoot que contiene el sniffer listo para capturar los paquetes. Usaremos un volumen compartido para guardar el .pcap para luego ser abierto con wireshark de Windows en modo grafico. Se debe ejecutar el nodo con permiso de administrador de red NET-ADMIN para que pueda tener acceso total al trafico de nuestra red, y la opción NET-RAW que permite escuchar todas las interfaces. Además, la interfaz eth0 no es la interfaz física de la máquina, sino la interfaz de red de todos los contenedores. Por último, el comando me permitira guardar el archivo en el volumen compartido bajo el nombre de trafico-ros2.pcap.

```
# Creamos nuestra red personalizada de Docker

docker network create proyecto

# Agregamos los contenedores del proyecto a la red. En este ejemplo tenemos
    un solo contenedor "sensor1" dentro del cual tenemos nuestros dos
    nodos sensor_node y reader_node.

docker network connect proyecto sensor1
```

```
# Ejecute el Docker nicolaka/netshoot con privilegios de administraci n de
     red al contenedor (--cap-add=NET_ADMIN) y con permisos para
    operaciones de bajo nivel en red, como capturar paquetes (--cap-add=
    NET_RAW). El comando que se ejecuta en el contenedor es tcpdump -i eth0
    -w /pcap/trafico_ros2.pcap que almacena el .pcap.

docker run --network proyecto -v C:\Users\pablo\Documents:/pcap --cap-add=
    NET_ADMIN --cap-add=NET_RAW nicolaka/netshoot tcpdump -i eth0 -w /pcap/
    trafico_ros2.pcap
```

En este escenario `eth0` en el sniffer sí ve todo lo que pasa entre sensor-node y reader-node, no necesitas tocar loopback ni interfaces extra ya que Docker enruta todo por el bridge virtual.

Como resultado podremos visualizar en el contenedor del sniff el siguente mensaje:

```
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length
    262144 bytes

# el contenedor que en modo ejucucion cuando creamos que ya tenemos los
    paquetes necesarios podemos terminar el contenedor y nos mostrara el
    mumero de paquetes adquiridos , como por ejemplo:

21 packets captured
22 packets received by filter
0 packets dropped by kernel
```

Ahora podremos ver en nuestro volumen compartido `C:` el archivo pcap que podremos abrir ya con Wireshark. Recordar que ROS usa el protocolo DDS sobre RTPS para comunicarse entre nodos, por lo que los puertos que deberemos monitorear son del 7400 al 7500 por lo tanto, dentro de wireshark podemos ejecutar:

```
udp.port >= 7400 && udp.port <= 7500
```

Reporte en el informe

Investigue y explique en el informe cómo funciona DDS (Data Distribution Service) sobre RTPS (Real-Time Publish-Subscribe protocol), incluyendo el propósito de filtrar los puertos 7400 y 7500. Además, describa los tipos de mensajes que se pueden observar, como por ejemplo los de handshake, y explique su función dentro del protocolo. En base a lo obtenido en la captura de Wireshark, explique los protocolos usados en cada capa del modelo TCP/IP.

Se recomienda ejecutar primero el Sniffer en el orden de ejecución de los nodos, ya que captura todo el tráfico desde que los nodos ROS2 comienzan a comunicarse y es ideal para un análisis completo, como el handshake DDS, discovery, etc. (ver Fig.3.3)

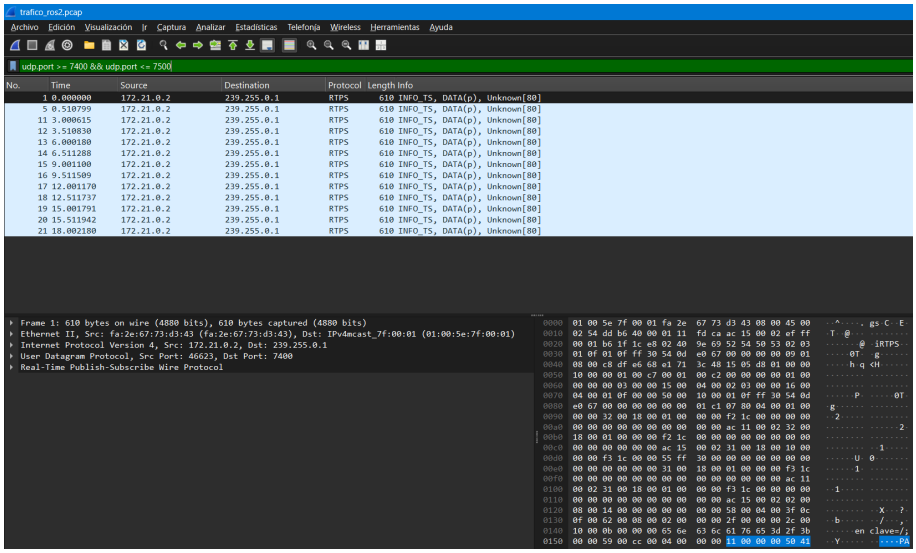

Figure 3.3: Análisis en Wireshark

3.7 Actividad Reto

Reporte en el informe

Tomando como base el ejemplo anterior con arduino ROS2 y docker , desarrolle la lectura del sensor de temperatura (potenciometro o sensor real) a traves de arduino, transforme la información en grados y muestre la información en Grafana. Investigue como usar Grafana.

Reporte la configuración de los siguientes desafíos del reto:

1. **Verificar todos los nodos funcionando**: Todos los nodos deben estar ejecutandose y mostrando la información de temperatura.
2. Habilitar QoS en la comunicación entre nodos y verificar el cambio en Wireshark.