EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Colima
División de Estudios de Posgrado e Investigación

Actividad: Práctica Multicast

Tecnologías de Internet

MAESTRÍA EN SISTEMAS COMPUTACIONALES
SEMESTRES AGOSTO 2022 – ENERO 2023

AUTORES:
Romero Velazco Fátima Adriana, Vazquez Velazquez Marco Dayan

PROFESORA:
Dra. Patricia Elizabeth Figueroa Millán

VILLA DE ÁLVAREZ, COLIMA. Noviembre de 2022.

Título de la práctica: PRÁCTICA MULTICAST

Objetivo

Realizar la entrega de datos de forma simultánea a un grupo de nodos receptores como destino, desde un emisor como origen vía Multicast para encender o apagar un diodo led desde un arduino con la lectura por puerto serial de un LDR desde otro arduino dentro de la misma red.

Marco teórico

Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML). [1]

Arduino es una plataforma de creación de electrónica de código abierto, la cual está basada en hardware y software libre, flexible y fácil de utilizar para los creadores y desarrolladores. Esta plataforma permite crear diferentes tipos de microordenadores de una sola placa a los que la comunidad de creadores puede darles diferentes tipos de uso. [2]

Metodología Utilizada

Se utilizaron 4 programas para la elaboración de esta práctica 2 en Python para el emisor y receptor y dos en Arduino, uno para la lectura del LDR y otro para encender o apagar el diodo LED según la sentencia recibida a cumplir del emisor del grupo multicast.

Módulo Emisor

Diseñar en Arduino IDE el programa que dará lectura vía puerto serial al valor analógico obtenido del Diodo LDR; en este programa se crea la condición necesaria para enviar el carácter “d” o “n” dependiendo del valor del umbral programado para cumplir la sentencia.

En Python se crea el programa para poder iniciar el proceso de detección de luz, con todas las funciones necesarias para la obtención de los datos **d** y **n**. Donde transmite la lectura al módulo receptor.

Módulo Receptor

El receptor es el que recibe las instrucciones del emisor y hace que el diodo led encienda o apague dependiendo las sentencias que reciba, ya sea **d** que es de día y **n** que es de noche.

Materiales Utilizados

- Programa Python.
- Visual Studio Code.
- IDE Arduino.
- Arduino UNO y Arduino YUN.
- Protoboard.
- Diodo LED.
- Resistencia 10kΩ y 330Ω
- Diodo LDR.
- Jumpers.
- Cable MicroUSB-USB / USB type A-USB type B.

Desarrollo

En el desarrollo de esta práctica se realizó la conexión del circuito de la figura 1 [3] en el arduino YUN.

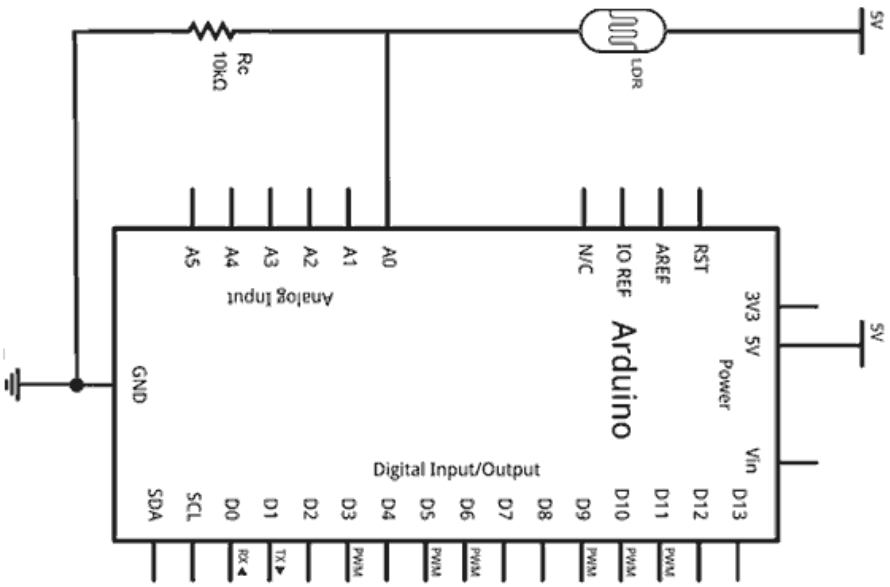


Figura 1. Esquema eléctrico conexión del LDR a arduino YUN.

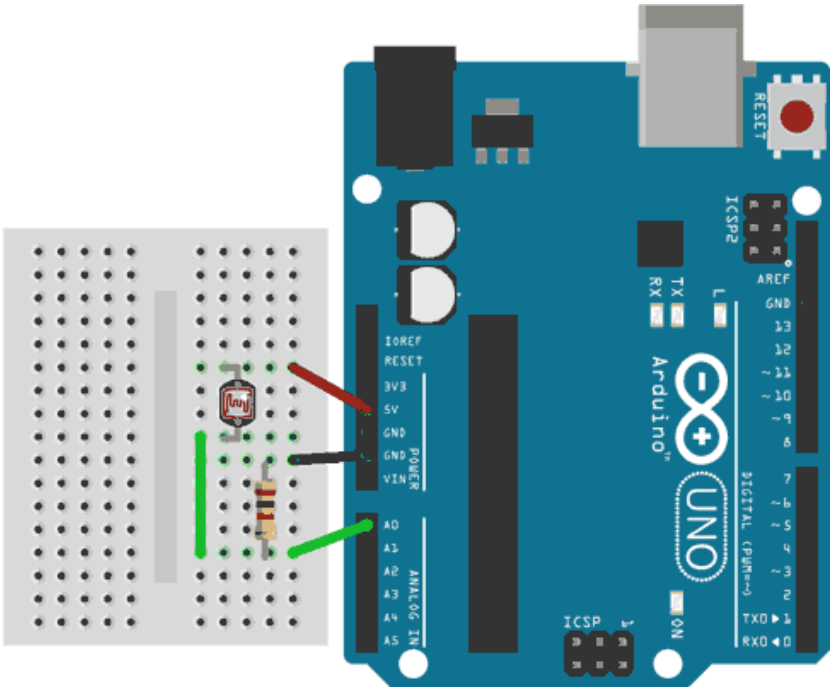


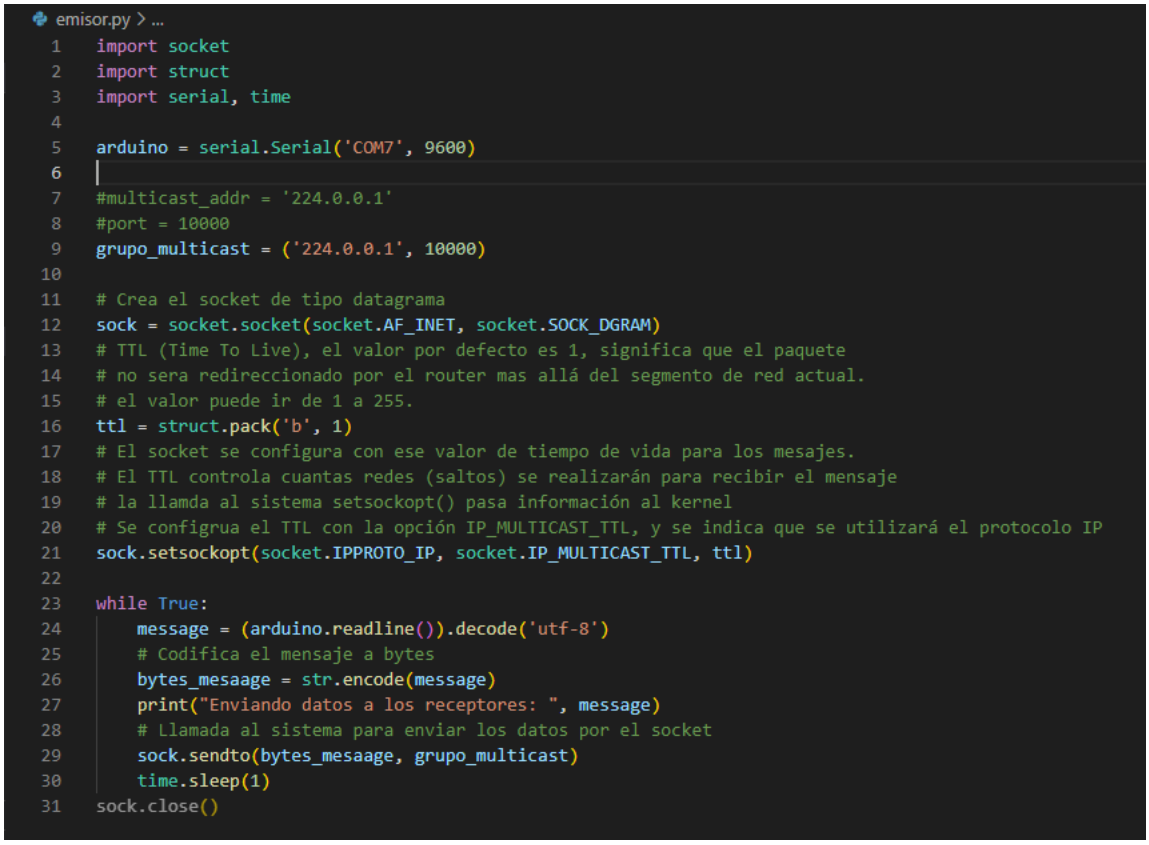
Figura 2. Vista conexión en protoboard del LDR a arduino YUN.

En Arduino IDE (figura 3) se crea el programa en el cual se leerá el puerto serial por el pin analógico A0; estos valores corresponden a la lectura del diodo LDR. Se declara en la línea 1 la asignación del pin A0 como “pinAnalogico”, de la línea 4 a 7 se crea la función “void setup()” donde se inicia la lectura del puerto serial a 9,600 baudios “Serial.begin(9600)”, finalmente de la línea 9 a la 25 se crea la función void loop() en la cual se declara una variable local tipo entero (valor) a la cual se le asignará el valor leído en “pinAnalogico”; seguido se crea la sentencia if eligiendo un umbral de 500; si “valor” es mayor que el umbral se manda a imprimir por puerto serial el carácter “d” que hace referencia a estado “día”, de lo contrario si “valor” no supera al umbral, se manda a imprimir el carácter “n” que hace referencia a estado “noche”. Los caracteres “n” y “d” serán recogidos por el programa del emisor creado en Python.

```
1   const byte pinAnalogico = A0;   //Pin del LDR
2
3
4   void setup()
5   {
6     Serial.begin(9600);
7   }
8
9   void loop()
10  {
11    // Leer pin analogico
12    int valor = analogRead(pinAnalogico);
13
14    // Mostrar valor pin analógico
15    //Serial.println(valor);
16    // Sentencia if con Arduino
17    if (valor > 500) {
18      Serial.println("d");
19    }else{
20      Serial.println("n");
21    }
22
23    // Retardo de 1000 ms
24    delay(1000);
25  }
```

Figura 3. Programa Arduino lectura LDR y escritura vía puerto serial.

En Python 3.1 (figura 4) se crea el primer programa que recibirá los caracteres de arduino con el programa de lectura del LDR enviados por el puerto serial; y que se transmitirán vía Multicast mediante protocolo IP a otra instancia dentro del mismo grupo multicast. En las líneas de 1 a 3 se importan las librerías correspondientes para importar las funciones: socket, struct, serial y time. En la línea 5 se declara la variable “arduino” para dar lectura a los datos enviados por puerto serial desde arduino especificando el #COM y velocidad de transmisión (9,600 baudios). En la línea 9 se declara la dirección del grupo multicast (224.0.0.1) y el puerto utilizado (10000) y se asigna a la variable “grupo_multicast”. En la línea 12 se crea el socket de tipo datagrama utilizando AF_INET ya que se utiliza IPv4 y se asigna a la variable “sock”. En la línea 16 se declara el tiempo de vida o “saltos” que podrá tener el paquete enviado, en este caso el valor es de 1 que se refiere a que no será redireccionado más allá del segmento de la red utilizada. En la línea 21 se manda a llamar a la función “setsockopt” para pasar la información al kernel (sock), procesar la solicitud y ejecutar la respuesta enviada; se configura el kernel con la opción IP_MULTICAST_TTL, se le asigna el valor de “ttl” e indica el uso del protocolo IP mediante IPPROTO_IP. Finalmente de la línea 23 a 30 se crea un ciclo while infinito. En este ciclo, en la línea 24 se declara la variable “message” y se le asigna el valor de la variable “arduino” que es la que contiene los caracteres de la lectura serial de arduino, se aplica el formato “utf-8” para la visualización en la consola una vez se mande a imprimir en la línea 27. En la línea 26 se codifica el valor de la variable “message” indicado que es de tipo string (cadena) y se asigna a la variable “bytes_mesaage”. Para finalizar el ciclo while en la línea 29 se envía la tupla “(bytes_mesaage, grupo_multicast)” mediante “sock.sendto” que contiene el byte del carácter “d” o “n” según corresponda y la dirección del grupo multicast, así como el puerto utilizado, respectivamente. Se da un tiempo de espera de 1 segundo entre cada mensaje enviado en la línea 30 y se cierra el socket utilizado.



```python
import socket
import struct
import serial, time

arduino = serial.Serial('COM7', 9600)

#multicast_addr = '224.0.0.1'
#port = 10000
grupo_multicast = ('224.0.0.1', 10000)

# Crea el socket de tipo datagrama
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# TTL (Time To Live), el valor por defecto es 1, significa que el paquete
# no sera redireccionado por el router mas allá del segmento de red actual.
# el valor puede ir de 1 a 255.
ttl = struct.pack('b', 1)
# El socket se configura con ese valor de tiempo de vida para los mesajes.
# El TTL controla cuantas redes (saltos) se realizarán para recibir el mensaje
# la llamda al sistema setsockopt() pasa información al kernel
# Se configrua el TTL con la opción IP_MULTICAST_TTL, y se indica que se utilizará el protocolo IP
sock.setsockopt(socket.IPPROTO_IP, socket.IP_MULTICAST_TTL, ttl)

while True:
    message = (arduino.readline()).decode('utf-8')
    # Codifica el mensaje a bytes
    bytes_mesaage = str.encode(message)
    print("Enviando datos a los receptores: ", message)
    # Llamada al sistema para enviar los datos por el socket
    sock.sendto(bytes_mesaage, grupo_multicast)
    time.sleep(1)
sock.close()
```

Figura 4. Programa lectura LDR y escritura de condiciones.

En el segundo programa de Python (figura 5) se crea el programa del receptor el cual tiene similitud con el del emisor. En las líneas 1 a 3 se inician las librerías. En la línea 5 se declara la variable “arduino” donde se inicializa el puerto serial hacia arduino con “serial.Serial” agregando el #COM y la velocidad de transferencia (9,600). En las líneas de 7 a 9 se define la dirección del grupo multicast (224.0.0.1), “bind_addr” donde se especificará la dirección IP o el nombre del host al que está vinculado el controlador del protocolo (0.0.0.0) y el puerto a utilizar (10000). En la línea 11 se crea el socket de tipo datagrama utilizando AF_INET ya que se utiliza IPv4 y se asigna a la variable “sock”. En la línea 12 con la función “inet_aton” de la librería socket las variables “multicast_addr” y “bind_addr” que contienen las direcciónes de la cadena de caracteres son convertidas de la notación estándar (0.0.0.0) a la representación binaria en orden de bytes de red y lo guarda en la estructura que tendrá la dirección convertida llamada “membership” [4]. En las líneas 14 y 15 se configura el comportamiento del socket mediante setsockopt, con “IPPROTO_IP” se configura para enviar o recibir datos

sin procesar para protocolos basados en IPv4 como TCP o UDP [5], mediante “IP_ADD_MEMBERSHIP” se une al grupo multicast IPv4 dentro de la interfaz IPv4 local guardada en la variable “membership” de la línea 12 [6]. Con “SOL_SOCKET y SO_REUSEADDR” se especifica para recuperar o establecer una opción dentro del mismo socket, en este caso se reutiliza la misma dirección en 1 (ON,TRUE), se pasa el nivel “sol_socket” y el valor que se desea tener, con esto se establecerá “so_reuseaddr” en el socket creado a 1 para reutilizar la misma dirección en el grupo multicast. En la línea 17 se asocia el socket a la dirección local especificada y el puerto asignado en la misma dirección. Finalmente de las líneas 19 a 24 se crea el ciclo while infinito; en la línea 20 con “recvfrom” se devuelve el objeto de bytes leído desde el socket y la dirección del socket del cliente como una tupla. En la línea 22 se manda a imprimir en consola el valor de la variable “mensaje” aplicado el formato “utf-8” el cual contiene el carácter “d” o “n” recibido del programa emisor en Python. En la línea 23 se manda a escribir el valor de “message” hacia el arduino UNO donde se tendrá la condición de encendido o apagado del diodo LED de acuerdo a las lecturas tomadas por el LDR del arduino YUN dando un time.sleep de 1 segundo.

receptor.py > ...

```python
import socket
import struct
import serial, time

arduino = serial.Serial('COM6', 9600)

multicast_addr = '224.0.0.1'
bind_addr = '0.0.0.0'
port = 10000

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
membership = socket.inet_aton(multicast_addr) + socket.inet_aton(bind_addr)

sock.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP, membership)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)

sock.bind((bind_addr, port))

while True:
    message, address = sock.recvfrom(255)
    mensaje = message.decode('utf-8')
    print("Recibiendo datos del emisor: ", mensaje)
    arduino.write(message)
    time.sleep(1)

```

Figura 5. Programa receptor y escritura de condiciones D y N hacia Arduino UNO.

Para finalizar, en el segundo programa de arduino (figura 6) se declara en la línea 1 y 2 la constante “pinLED” del pin 9 del arduino UNO, así como una bandera nombrada “estado” que se utilizará para las condiciones de encendido y apagado. En la función void setup() de la línea 4 a 8 se inicializa el puerto serial a 9,600 baudios y se declara “pinLED” como salida. En la función void loop(), se declara en la línea 12 la variable “option” de tipo char para almacenar el carácter leído por el puerto serial. En la línea 12 se inicia la condición if, donde si la variable “option” tomaba el valor de “n” y estado es igual a 0, se mandaría un estado “HIGH” al pinLED (pin 9) de arduino, la variable “estado” ahora tomaría el valor de 1 y encenderá el diodo LED, de lo contrario si el valor recibido en la variable “option” es “d” y como ahora el valor de “estado” es 1, la sentencia mandará un estado LOW al pinLED y apagará el diodo LED, donde ahora el valor de la bandera “estado” cambiará a 0 y estará a la espera de recibir nuevas lecturas.

LED.ino

```
1   const int pinLED = 9;
2   int estado=0;
3
4   void setup()
5   {
6     Serial.begin(9600);
7     pinMode(pinLED, OUTPUT);
8   }
9
10  void loop()
11  {
12    char option = Serial.read();
13      if (option == 'n' && estado==0)
14      {
15        digitalWrite(pinLED, HIGH);
16        delay(100);
17        estado=1;
18      }
19      if (option == 'd' && estado==1)
20      {
21        digitalWrite(pinLED, LOW);
22        delay(100);
23        estado=0;
24            }
25  }
```

Figura 6. Programa en arduino, encender o apagar el led de acuerdo a la lectura de puerto serial.

Resultados

Se construye el circuito del segundo arduino (figura 7) donde se utiliza el pin 9 como salida.

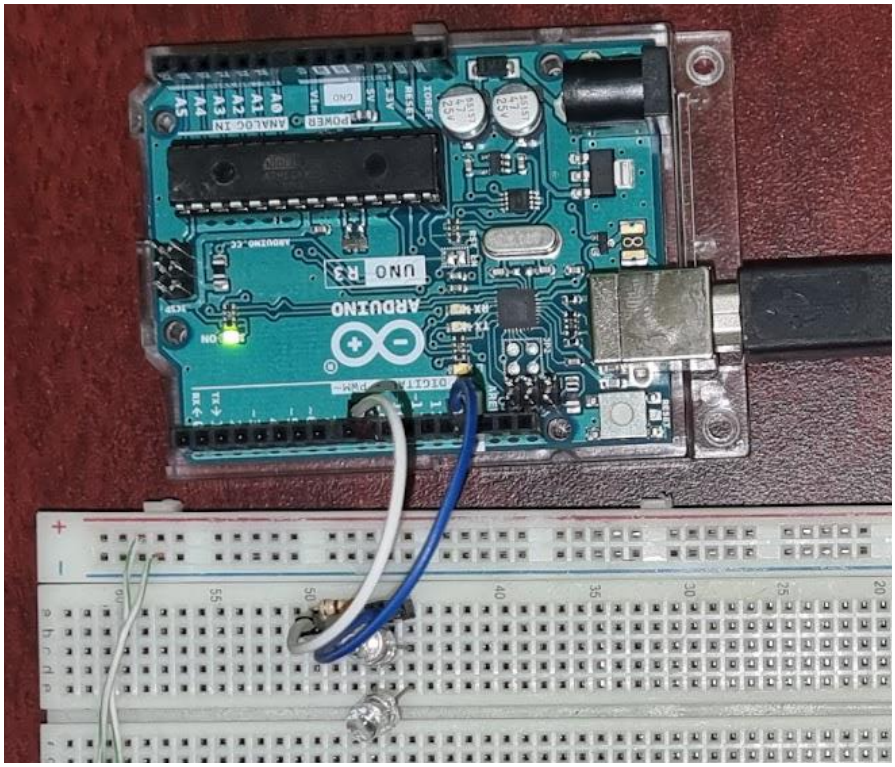


Figura 7. Conexión arduino UNO para el control del diodo LED.

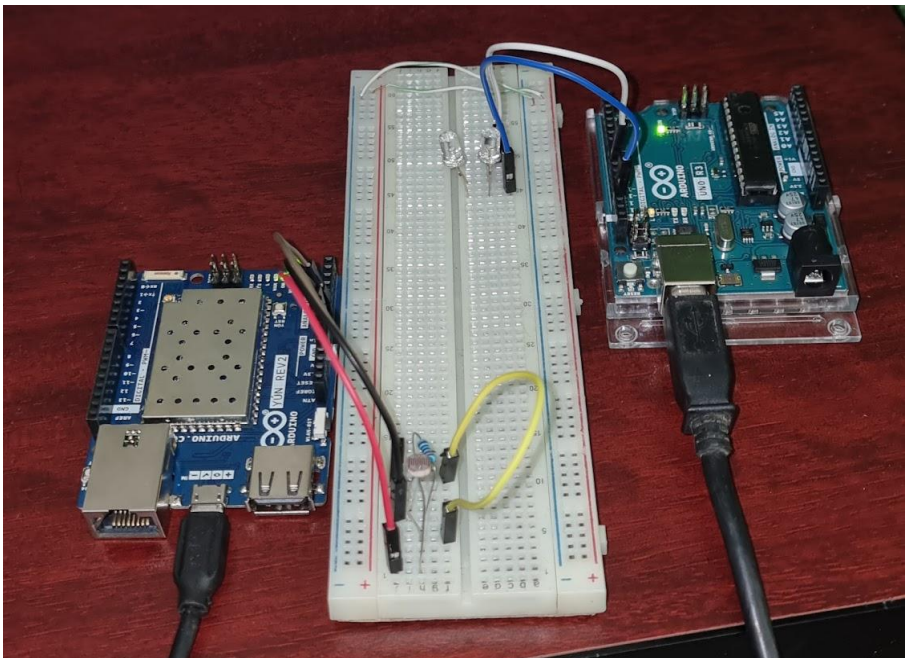


Figura 8. Vista de ambas conexiones arduino UNO y YUN.

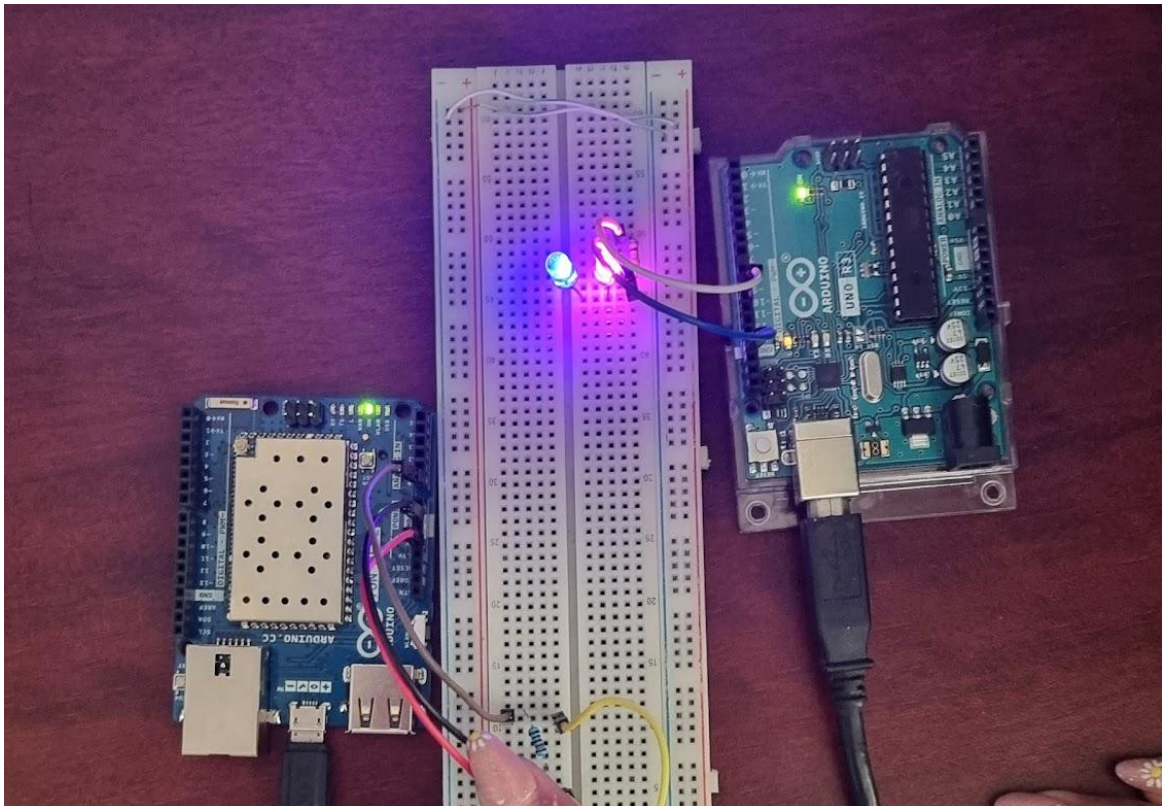
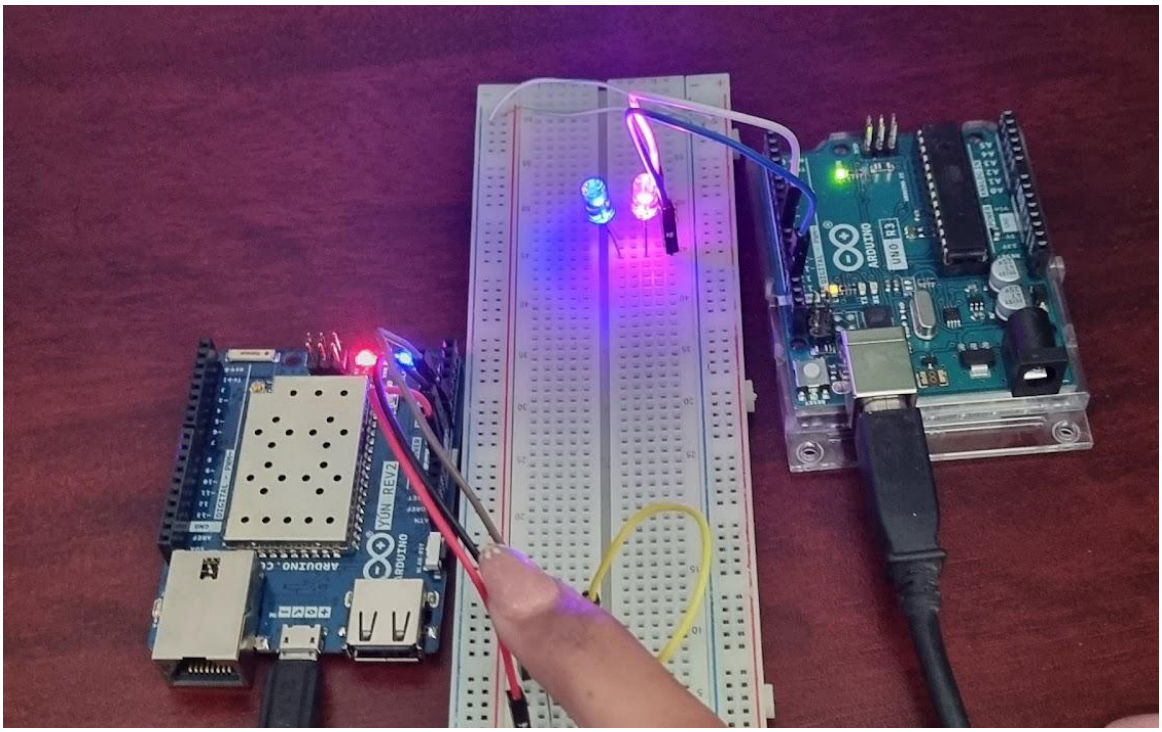


Figura 9 y 10. Privando de luz al LDR se manda el carácter “n” y se enciende el LED.



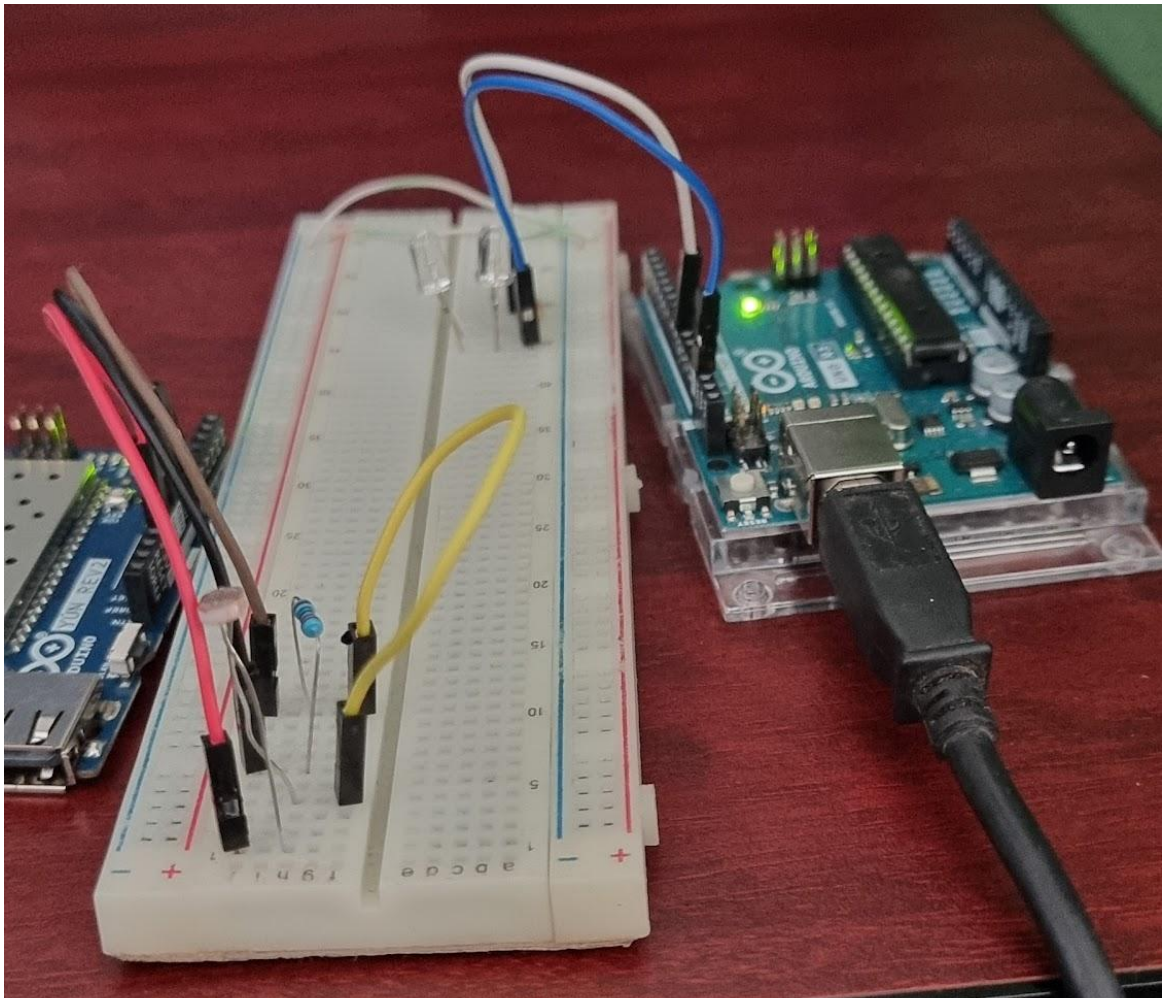


Figura 11. Si el LDR recibe la suficiente luz se manda el carácter “d” y se apaga el LED.

Conclusiones

Al realizar esta práctica, nos dimos cuenta que para tener una buena comunicación de datos, es necesario tener una buena conexión a la red que utilizamos, ya que con ellos podemos hacer que el emisor no tenga problema alguno con enviar datos que el receptor comunique.

Algo importante que se tiene que tener en cuenta es que el receptor y emisor tenga los mismos datos con los que se comunique, es decir, utilicen el mismo puerto con el que se conecta al arduino, para que la comunicación multicast con otros equipos puedan ser buena conectando a la misma red.

Como sabemos la práctica desarrollada en esta materia muestra y representa lo que es el internet de las cosas, en este caso la práctica con la cual hicimos el circuito de eventos con el cual se toma la decisión, de encender el foco si no hay luz o si hay luz que se apague.

Se podría llegar a utilizar esta práctica a mayor escala, solo sería cuestión de agregar los demás materiales y se podría formar.

Referencias

[1] *"¿Qué es Python? | Guía de Python para principiantes de la nube | AWS". Amazon Web Services, Inc. https://aws.amazon.com/es/what-is/python/ (accedido el 18 de noviembre de 2022).*

[2] Llamas, L. (16 de marzo de 2015). *Medir nivel de luz con Arduino y fotoresistencia LDR (GL55)*. Luis Llamas. https://www.luisllamas.es/medir-nivel-luz-con-arduino-y-fotoresistencia-ldr/

[3] *PySerial Python Arduino comunicación serial*. (s. f.). HETPRO/TUTORIALES. https://hetpro-store.com/TUTORIALES/pyserial-python-arduino-comunicacion-serial/

[4] Bonet, E. (s. f.). *Estructuras y funciones de programación de sockets.* informatica.uv.es. http://informatica.uv.es/iiguia/R/apuntes/laboratorio/Funciones.pdf

[5] *What is the difference between IPPROTO_IP and IPPROTO_RAW?* (10 de junio de 2014). Stack Overflow. https://stackoverflow.com/questions/24590818/what-is-the-difference-between-ipproto-ip-and-ipproto-raw

[6] Corporation, I. (3 de marzo de 2021). *IBM Documentation*. IBM - United States. https://www.ibm.com/docs/en/zos/2.3.0?topic=options-ip-add-membership-ip-drop-membership

[7] Setup Labs. (28 de septiembre de 2021). *Programación en PYTHON y Arduino para ENVIAR y RECIBIR Datos* [Video]. YouTube. https://www.youtube.com/watch?v=J0CMME2gd9I

[8] Llamas, L. (25 de enero de 2016). *Controlar Arduino con Python y la librería PySerial*. Luis Llamas. https://www.luisllamas.es/controlar-arduino-con-python-y-la-libreria-pyserial/

[9] *ENCENDER Y APAGAR CON UN PULSADOR - ARDUINO*. (s. f.). ARDUINO. https://arduino.micro-log.com/encender-y-apagar-con-un-pulsador/#:~:text=Si%20pulsas%20el%20pulsador%20y,variable%20de%20estado%20por%200