Universidad Santo Tomás

Laboratorio 3

David Esteban Diaz Castro
Ferney Arturo Amaya Gómez
Jhonny Alejandro Mejia

Octubre 5, 2025

Índice general

Capítulo 1

Introducción

Este documento presenta la implementación de diferentes topologías de red junto con la configuración de switches y comunicación con Arduino. Se incluyen configuraciones para topologías en estrella, árbol, anillo y malla, así como los códigos fuente en Python y Arduino correspondientes.

Capítulo 2

Configuración del Switch via Putty

2.1. Descripción

En esta sección se presenta la configuración del switch mediante Putty para conectar múltiples dispositivos en una red local.

2.2. Comandos para Configuración del Switch

```
enable
configure terminal
host-name David
interface vlan 1
ip address 192.168.80.1 255.255.255.0
no shutdown
exit
interface fastethernet 0/1
description PC1
switchport mode access
no shutdown
exit
interface fastethernet 0/2
description PC2
switchport mode access
no shutdown
exit
interface fastethernet 0/3
description PC3
switchport mode access
no shutdown
exit
interface fastethernet 0/4
description PC4
switchport mode access
no shutdown
exit
end
write memory
show ip interface brief
show interfaces status
show vlan
show mac-address-table
```

2.3. Configuración de las PCs

2.3.1. PC1

```
netsh interface ip set address "Ethernet" static 192.168.80.101
    255.255.255.0 192.168.80.1
ping 192.168.80.102
ping 192.168.80.103
ping 192.168.80.104
ping 192.168.80.1
```

2.3.2. PC2

```
netsh interface ip set address "Ethernet" static 192.168.80.102
    255.255.255.0 192.168.80.1
ping 192.168.80.101
ping 192.168.80.103
ping 192.168.80.104
ping 192.168.80.1
```

2.3.3. PC3

```
netsh interface ip set address "Ethernet" static 192.168.80.103
    255.255.255.0 192.168.80.1
ping 192.168.80.101
ping 192.168.80.102
ping 192.168.80.104
ping 192.168.80.1
```

2.3.4. PC4

```
netsh interface ip set address "Ethernet" static 192.168.80.104
    255.255.255.0 192.168.80.1
ping 192.168.80.101
ping 192.168.80.102
ping 192.168.80.103
ping 192.168.80.1
```

2.4. Comandos de Verificación

```
show running-config
show ip interface brief
show interfaces status
show mac-address-table
ping 192.168.80.101
```

```
ping 192.168.80.102
ping 192.168.80.103
ping 192.168.80.104
```

Capítulo 3

Topología en Estrella

3.1. Descripción

La topología en estrella consiste en un servidor central que gestiona todas las comunicaciones entre los nodos clientes. El Arduino está conectado al servidor central.

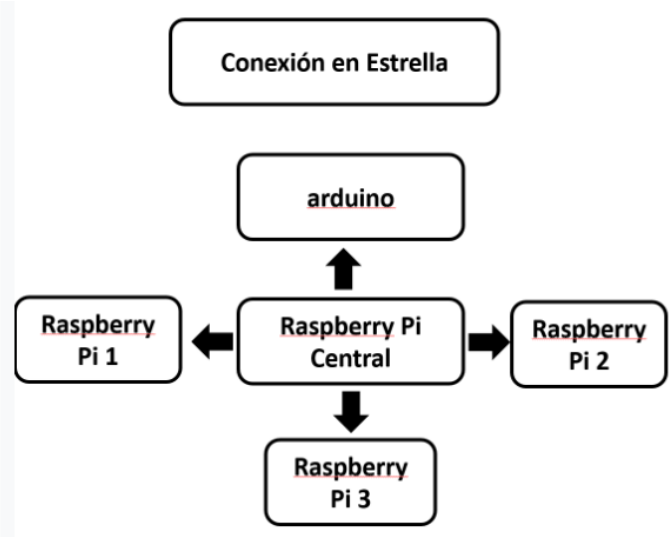


Figura 3.1: Diagrama de la topología en estrella

3.2. Resultados

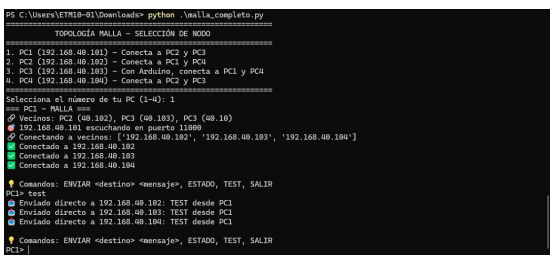


Figura 3.2: Resultado 1 - Topología Estrella

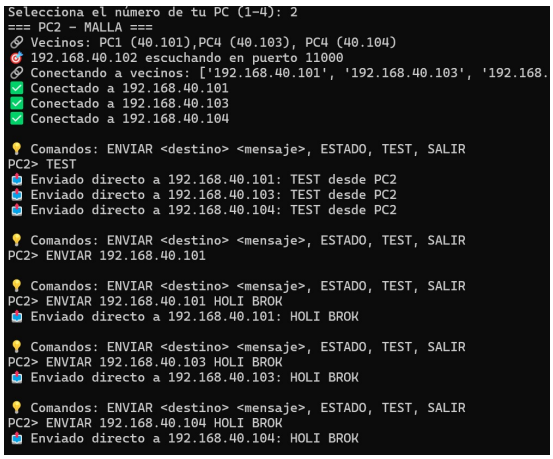


Figura 3.3: Resultado 2 - Topología Estrella

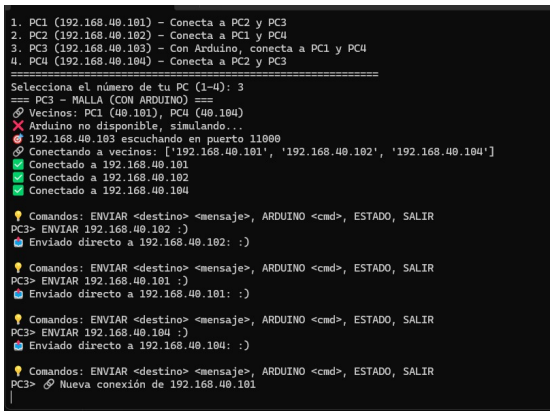


Figura 3.4: Resultado 3 - Topología Estrella

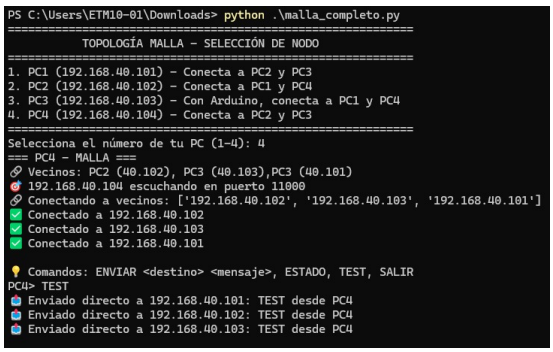


Figura 3.5: Resultado 4 - Topología Estrella

3.3. Código Fuente en Python

```
import socket
import threading
import time
import json
import serial
```

```
6  import uuid
7
8  class ServidorEstrella:
9      def __init__(self, ip_servidor):
10         self.ip = ip_servidor
11         self.puerto = 11000
12         self.clientes_conectados = {}
13         self.mensajes_recibidos = set()
14         self.arduino = None
15
16     def conectar_arduino(self):
17         try:
18             self.arduino = serial.Serial('COM3', 9600, timeout=1)
19             time.sleep(2)
20             print("    Arduino conectado al servidor central")
21             return True
22         except:
23             print("    Arduino no disponible en el servidor central")
24             return False
25
26     def manejar_cliente(self, cliente, addr):
27         ip_cliente = addr[0]
28         print(f"    Cliente conectado: {ip_cliente}")
29         self.clientes_conectados[ip_cliente] = cliente
30
31         try:
32             while True:
33                 datos = cliente.recv(2048).decode()
34                 if not datos:
35                     break
36
37                 try:
38                     mensaje = json.loads(datos)
39                 except:
40                     continue
41
42                 if mensaje['id'] in self.mensajes_recibidos:
43                     continue
44                 self.mensajes_recibidos.add(mensaje['id'])
45
46                 print(f"       [{self.ip}] De {mensaje['origen']} para {
                       mensaje['destino']}: {mensaje['mensaje']}")
47
48                 if mensaje['destino'] == self.ip and mensaje['mensaje'].
                       startswith("ARDUINO:"):
49                     self.procesar_comando_arduino(mensaje['mensaje'],
                           mensaje['origen'])
50                 else:
51                     self.reenviar_mensaje(mensaje['destino'], datos)
52
53         except Exception as e:
54             print(f"    Error con cliente {ip_cliente}: {e}")
55         finally:
56             self.eliminar_cliente(ip_cliente)
57             cliente.close()
58
59     def procesar_comando_arduino(self, mensaje, origen):
60         comando = mensaje.split(":")[1].strip().upper()
```

```
61          print(f"       Comando Arduino desde {origen}: {comando}")
62
63          if comando in ["ON", "OFF"]:
64              if self.arduino:
65                  self.arduino.write(f"{comando}\n".encode())
66                  print(f"    Arduino ejecut  {comando}")
67                  self.enviar_a_cliente(origen, f"ACK: Arduino {comando}
                        ejecutado")
68              else:
69                  print(f"       Simulaci n Arduino {comando}")
70                  self.enviar_a_cliente(origen, f"SIM: Arduino {comando} (
                        simulado)")
71          else:
72              self.enviar_a_cliente(origen, f"ERROR: Comando no v lido: {
                    comando}")
73
74      def reenviar_mensaje(self, destino, mensaje_json):
75          if destino in self.clientes_conectados:
76              try:
77                  self.clientes_conectados[destino].send(mensaje_json.
                        encode())
78                  print(f"       Mensaje reenviado a {destino}")
79              except Exception as e:
80                  print(f"    Error reenviando a {destino}: {e}")
81                  self.eliminar_cliente(destino)
82          else:
83              print(f"    Destino {destino} no conectado")
84
85      def enviar_a_cliente(self, destino, mensaje):
86          if destino in self.clientes_conectados:
87              mensaje_completo = json.dumps({
88                  'id': str(uuid.uuid4()),
89                  'origen': self.ip,
90                  'destino': destino,
91                  'mensaje': mensaje,
92                  'timestamp': time.time()
93              })
94              try:
95                  self.clientes_conectados[destino].send(mensaje_completo.
                        encode())
96              except:
97                  self.eliminar_cliente(destino)
98
99      def eliminar_cliente(self, ip_cliente):
100         if ip_cliente in self.clientes_conectados:
101             del self.clientes_conectados[ip_cliente]
102             print(f"       Cliente desconectado: {ip_cliente}")
103
104     def mostrar_estado(self):
105         print(f"\  n      ESTADO SERVIDOR {self.ip}")
106         print(f"       Clientes conectados: {list(self.
                clientes_conectados.keys())}")
107         print(f"       Arduino: {'Conectado' if self.arduino else 'No
                disponible'}")
108
109     def iniciar_servidor(self):
110         servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
111         servidor.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
```

```
112         servidor.bind((self.ip, self.puerto))
113         servidor.listen(10)
114 
115         print(f"       SERVIDOR ESTRELLA {self.ip} escuchando en puerto
                {self.puerto}")
116 
117         while True:
118             try:
119                 cliente, addr = servidor.accept()
120                 threading.Thread(target=self.manejar_cliente, args=(
                        cliente, addr), daemon=True).start()
121             except Exception as e:
122                 print(f"    Error aceptando conexi n: {e}")
123 
124 class ClienteEstrella:
125     def __init__(self, ip_propia, ip_servidor):
126         self.ip = ip_propia
127         self.ip_servidor = ip_servidor
128         self.puerto = 11000
129         self.servidor = None
130         self.conectado = False
131 
132     def conectar_servidor(self):
133         try:
134             self.servidor = socket.socket(socket.AF_INET, socket.
                    SOCK_STREAM)
135             self.servidor.settimeout(5)
136             self.servidor.connect((self.ip_servidor, self.puerto))
137             self.conectado = True
138             print(f"    Conectado al servidor central {self.ip_servidor}
                    ")
139 
140             threading.Thread(target=self.escuchar_servidor, daemon=True)
                    .start()
141             return True
142         except Exception as e:
143             print(f"    No se pudo conectar al servidor: {e}")
144             return False
145 
146     def escuchar_servidor(self):
147         while self.conectado:
148             try:
149                 datos = self.servidor.recv(2048).decode()
150                 if not datos:
151                     self.conectado = False
152                     break
153 
154                 try:
155                     mensaje = json.loads(datos)
156                     self.procesar_mensaje(mensaje)
157                 except:
158                     continue
159 
160             except socket.timeout:
161                 continue
162             except:
163                 self.conectado = False
164                 break
```

```
165
166     def procesar_mensaje(self, mensaje):
167         print(f"       [{self.ip}] De {mensaje['origen']}: {mensaje['
                mensaje']}")
168
169     def enviar_mensaje(self, destino, mensaje):
170         if not self.conectado:
171             print("    No conectado al servidor")
172             return False
173
174         mensaje_completo = json.dumps({
175             'id': str(uuid.uuid4()),
176             'origen': self.ip,
177             'destino': destino,
178             'mensaje': mensaje,
179             'timestamp': time.time()
180         })
181
182         try:
183             self.servidor.send(mensaje_completo.encode())
184             print(f"       Mensaje enviado a {destino} via servidor")
185             return True
186         except Exception as e:
187             print(f"    Error enviando mensaje: {e}")
188             self.conectado = False
189             return False
190
191     def mostrar_estado(self):
192         print(f"\  n     ESTADO CLIENTE {self.ip}")
193         print(f"       Servidor: {'Conectado' if self.conectado else '
                Desconectado'}")
194
195     def cerrar_conexion(self):
196         self.conectado = False
197         if self.servidor:
198             self.servidor.close()
199
200 class PC1(ClienteEstrella):
201     def __init__(self):
202         super().__init__('192.168.40.101', '192.168.40.100')
203
204     def iniciar(self):
205         print("=== PC1 - CLIENTE ESTRELLA ===")
206         if self.conectar_servidor():
207             self.interfaz_usuario()
208
209     def interfaz_usuario(self):
210         while True:
211             entrada = input("PC1> ").strip()
212             if not entrada: continue
213             partes = entrada.split()
214             comando = partes[0].upper()
215
216             if comando == "SALIR":
217                 break
218             elif comando == "ESTADO":
219                 self.mostrar_estado()
220             elif comando == "ENVIAR" and len(partes) >= 3:
```

```
                destino = partes[1]
                mensaje = " ".join(partes[2:])
                self.enviar_mensaje(destino, mensaje)
            elif comando == "ARDUINO" and len(partes) >= 2:
                comando_arduino = " ".join(partes[1:])
                self.enviar_mensaje('192.168.40.100', f"ARDUINO:{
                    comando_arduino}")
        self.cerrar_conexion()

if __name__ == "__main__":
    print("=" * 60)
    print("            TOPOLOG A ESTRELLA - SELECCI N DE MODO")
    print("=" * 60)
    print("S. Servidor Central (192.168.40.100) - Con Arduino")
    print("1. PC1 (192.168.40.101)")
    print("2. PC2 (192.168.40.102)")
    print("3. PC3 (192.168.40.103)")
    print("4. PC4 (192.168.40.104)")
    print("=" * 60)

    while True:
        opcion = input("Selecciona modo (S/1/2/3/4): ").strip().upper()

        if opcion == "S":
            servidor = ServidorEstrella('192.168.40.100')
            servidor.conectar_arduino()
            servidor.iniciar_servidor()
            break
        elif opcion == "1":
            nodo = PC1()
            nodo.iniciar()
            break
        else:
            print("    Opci n no v lida")
```

Listing 3.1: Topología Estrella - Python

3.4. Código Fuente en Arduino

```
const int LED_PIN = 13;
int contador = 0;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
  delay(2000);
  Serial.println("ARDUINO ESTRELLA - COMANDOS: ON, OFF, STATUS");
}

void loop() {
  if (Serial.available() > 0) {
    String comando = Serial.readStringUntil('\n');
    comando.trim();
    comando.toUpperCase();

    Serial.print("Comando: ");
```

```
18      Serial.println(comando);
19
20      if (comando == "ON") {
21        digitalWrite(LED_PIN, HIGH);
22        Serial.println("LED ENCENDIDO");
23      }
24      else if (comando == "OFF") {
25        digitalWrite(LED_PIN, LOW);
26        Serial.println("LED APAGADO");
27      }
28      else if (comando == "STATUS") {
29        Serial.println("ARDUINO ACTIVO");
30      }
31      else {
32        Serial.println("COMANDO NO RECONOCIDO");
33      }
34    }
35    delay(100);
36  }
```

Listing 3.2: Topología Estrella - Arduino

Capítulo 4

Topología en Árbol

4.1. Descripción

La topología en árbol organiza los nodos de forma jerárquica, con un nodo raíz, nodos rama intermedios y nodos hoja finales.

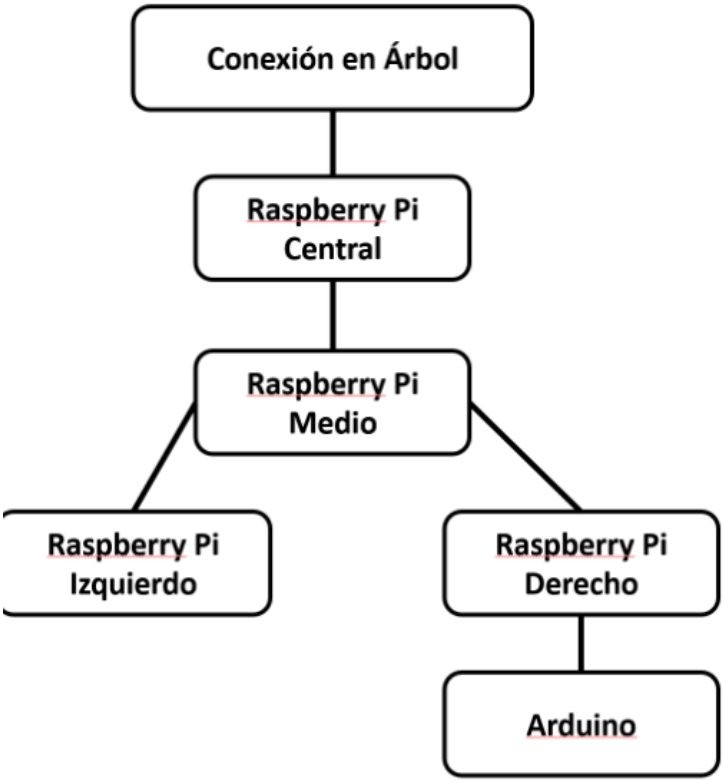


Figura 4.1: Diagrama de la topología en árbol

4.2. Resultados

```
PS C:\Users\ETM10-01\Downloads> python .\malla_completo.py
============================================================
           TOPOLOGÍA MALLA - SELECCIÓN DE NODO
============================================================
1. PC1 (192.168.40.101) - Conecta a PC2 y PC3
2. PC2 (192.168.40.102) - Conecta a PC1 y PC4
3. PC3 (192.168.40.103) - Con Arduino, conecta a PC1 y PC4
4. PC4 (192.168.40.104) - Conecta a PC2 y PC3
============================================================
Selecciona el número de tu PC (1-4): 1
=== PC1 - MALLA ===
🔗 Vecinos: PC2 (40.102), PC3 (40.103), PC3 (40.10)
🎯 192.168.40.101 escuchando en puerto 11000
🔗 Conectando a vecinos: ['192.168.40.102', '192.168.40.103', '192.168.40.104']
✅ Conectado a 192.168.40.102
✅ Conectado a 192.168.40.103
✅ Conectado a 192.168.40.104

💡 Comandos: ENVIAR <destino> <mensaje>, ESTADO, TEST, SALIR
PC1> test
📤 Enviado directo a 192.168.40.102: TEST desde PC1
📤 Enviado directo a 192.168.40.103: TEST desde PC1
📤 Enviado directo a 192.168.40.104: TEST desde PC1

💡 Comandos: ENVIAR <destino> <mensaje>, ESTADO, TEST, SALIR
PC1>
```

Figura 4.2: Resultado 1 - Topología Árbol

```
PS C:\Users\ETM10-01\Downloads> python .\malla_completo.py
============================================================
           TOPOLOGÍA MALLA - SELECCIÓN DE NODO
============================================================
1. PC1 (192.168.40.101) - Conecta a PC2 y PC3
2. PC2 (192.168.40.102) - Conecta a PC1 y PC4
3. PC3 (192.168.40.103) - Con Arduino, conecta a PC1 y PC4
4. PC4 (192.168.40.104) - Conecta a PC2 y PC3
============================================================
Selecciona el número de tu PC (1-4): 1
=== PC1 - MALLA ===
🔗 Vecinos: PC2 (40.102), PC3 (40.103), PC3 (40.10)
🎯 192.168.40.101 escuchando en puerto 11000
🔗 Conectando a vecinos: ['192.168.40.102', '192.168.40.103', '192.168.40.104']
✅ Conectado a 192.168.40.102
✅ Conectado a 192.168.40.103
✅ Conectado a 192.168.40.104

💡 Comandos: ENVIAR <destino> <mensaje>, ESTADO, TEST, SALIR
PC1> test
📤 Enviado directo a 192.168.40.102: TEST desde PC1
📤 Enviado directo a 192.168.40.103: TEST desde PC1
📤 Enviado directo a 192.168.40.104: TEST desde PC1

💡 Comandos: ENVIAR <destino> <mensaje>, ESTADO, TEST, SALIR
PC1>
```

Figura 4.3: Resultado 2 - Topología Árbol

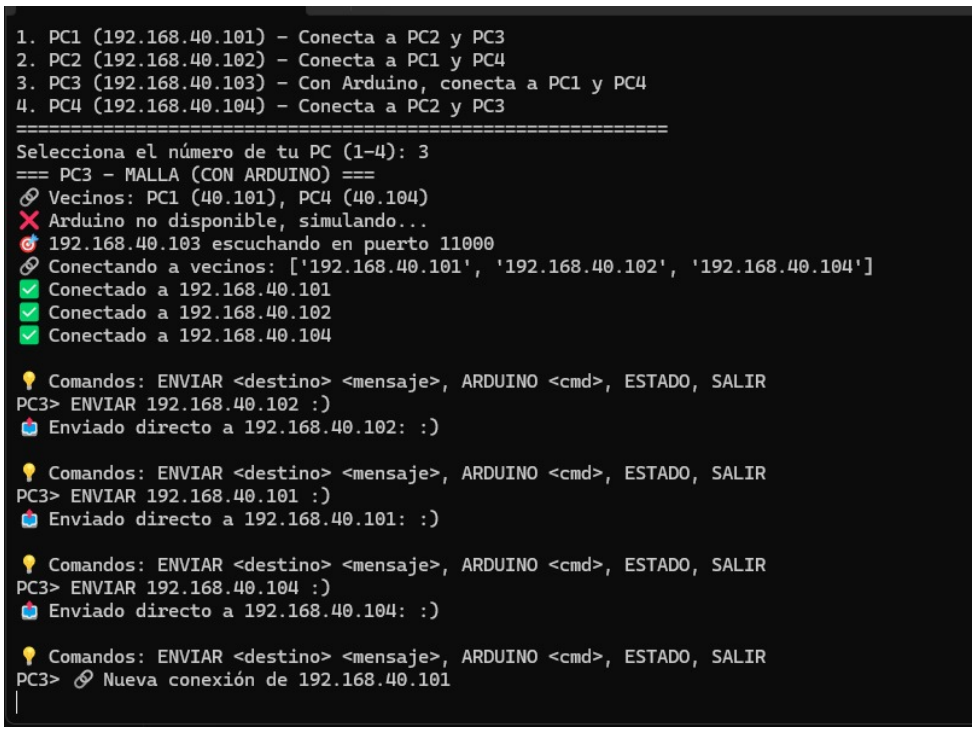


Figura 4.4: Resultado 3 - Topología Árbol

4.3. Conexiones

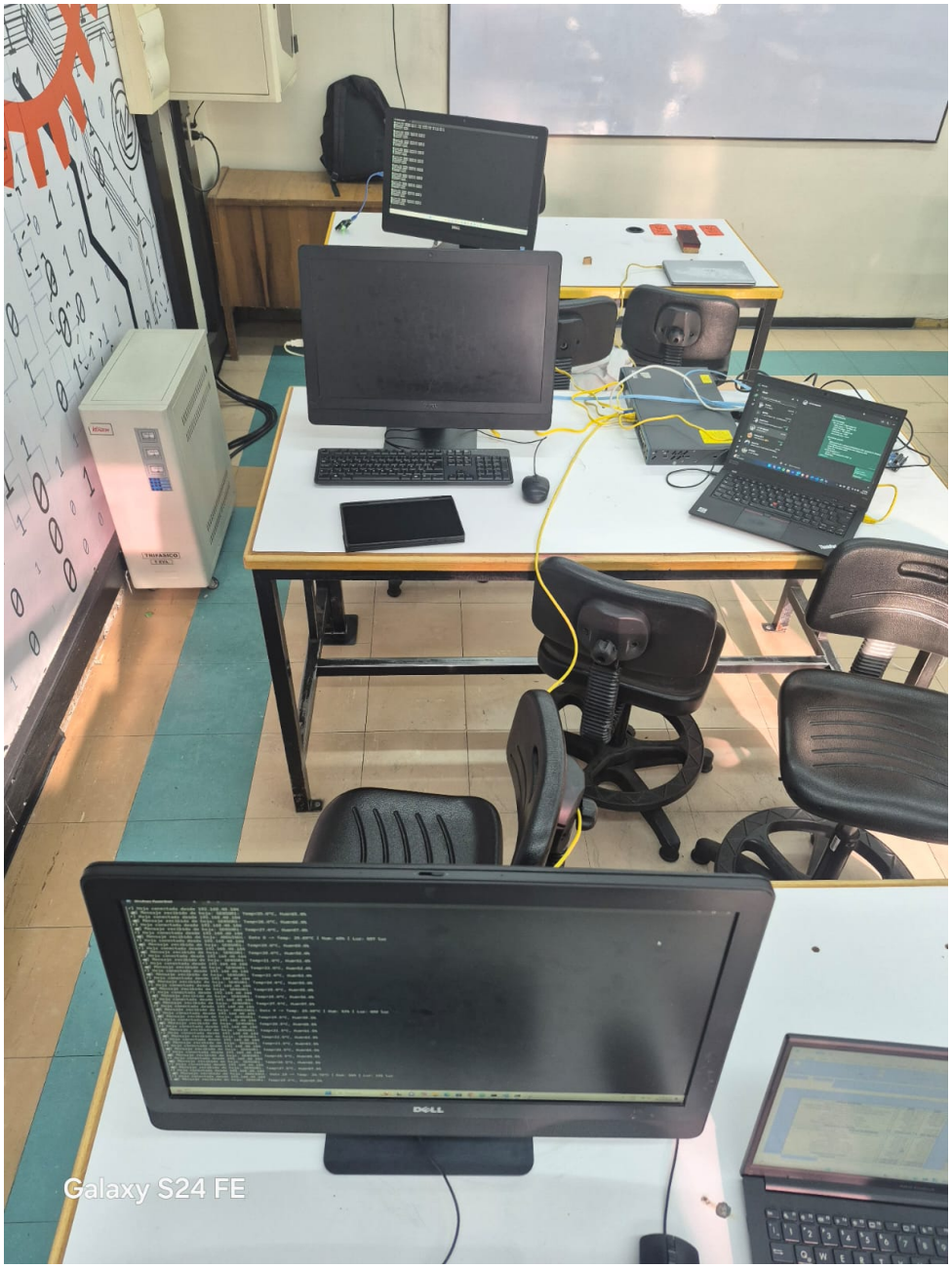


Figura 4.5: Diagrama de conexiones - Topología Árbol

4.4. Código Fuente en Python

```
import socket
import threading
import time
import json
import serial
import uuid

```

```
8  class NodoArbol:
9      def __init__(self, ip_propia, rol):
10         self.ip = ip_propia
11         self.rol = rol
12         self.padre = None
13         self.hijos = {}
14         self.puerto = 11000
15         self.arduino = None
16         self.mensajes_recibidos = set()
17         self.nivel = 0
18
19     def conectar_padre(self, ip_padre):
20         try:
21             sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
22             sock.settimeout(5)
23             sock.connect((ip_padre, self.puerto))
24             self.padre = sock
25             print(f"    Conectado al padre {ip_padre}")
26
27             mensaje_registro = json.dumps({
28                 'id': str(uuid.uuid4()),
29                 'origen': self.ip,
30                 'destino': ip_padre,
31                 'tipo': 'REGISTRO',
32                 'rol': self.rol,
33                 'mensaje': f"REGISTRO:{self.rol}:{self.ip}",
34                 'timestamp': time.time()
35             })
36             sock.send(mensaje_registro.encode())
37             return True
38         except Exception as e:
39             print(f"    No se pudo conectar al padre {ip_padre}: {e}")
40             return False
41
42     def aceptar_hijo(self, cliente, addr, datos_registro):
43         try:
44             ip_hijo = addr[0]
45             self.hijos[ip_hijo] = cliente
46             print(f"    Hijo registrado: {ip_hijo} como {datos_registro
                   ['rol']}")
47
48             threading.Thread(target=self.manejar_hijo, args=(cliente,
                   ip_hijo), daemon=True).start()
49         except Exception as e:
50             print(f"    Error registrando hijo: {e}")
51
52     def manejar_hijo(self, cliente, ip_hijo):
53         try:
54             while True:
55                 datos = cliente.recv(2048).decode()
56                 if not datos:
57                     break
58
59                 try:
60                     mensaje = json.loads(datos)
61                 except:
62                     continue
63
```

```
64                  if mensaje['id'] in self.mensajes_recibidos:
65                      continue
66                  self.mensajes_recibidos.add(mensaje['id'])
67
68                  print(f"       [{self.ip}] De hijo {ip_hijo} para {
                        mensaje['destino']}: {mensaje['mensaje']}")
69
70                  self.reenviar_mensaje_arbol(mensaje, ip_hijo)
71
72          except Exception as e:
73              print(f"    Error con hijo {ip_hijo}: {e}")
74          finally:
75              self.eliminar_hijo(ip_hijo)
76
77      def reenviar_mensaje_arbol(self, mensaje, origen_inmediato):
78          destino = mensaje['destino']
79
80          if destino == self.ip:
81              self.procesar_mensaje_local(mensaje['mensaje'], mensaje['
                    origen'])
82              return
83
84          if self.es_descendiente(destino):
85              for ip_hijo in self.hijos:
86                  if self.es_descendiente_de(destino, ip_hijo):
87                      self.enviar_a_hijo(ip_hijo, mensaje)
88                      break
89          else:
90              if self.padre:
91                  self.enviar_a_padre(mensaje)
92
93      def es_descendiente(self, ip_destino):
94          estructuras = {
95              '192.168.40.101': ['192.168.40.102', '192.168.40.103'],
96              '192.168.40.102': ['192.168.40.104', '192.168.40.105'],
97              '192.168.40.103': ['192.168.40.106', '192.168.40.107'],
98          }
99
100         for padre, hijos in estructuras.items():
101             if ip_destino in hijos and padre == self.ip:
102                 return True
103         return False
104
105     def es_descendiente_de(self, ip_destino, ip_hijo):
106         estructuras = {
107             '192.168.40.102': ['192.168.40.104', '192.168.40.105'],
108             '192.168.40.103': ['192.168.40.106', '192.168.40.107'],
109         }
110         return ip_hijo in estructuras and ip_destino in estructuras[
                ip_hijo]
111
112     def enviar_a_padre(self, mensaje):
113         if self.padre:
114             try:
115                 self.padre.send(json.dumps(mensaje).encode())
116             except:
117                 print("    Error enviando al padre")
118                 self.padre = None
```

```
119
120     def enviar_a_hijo(self, ip_hijo, mensaje):
121         if ip_hijo in self.hijos:
122             try:
123                 self.hijos[ip_hijo].send(json.dumps(mensaje).encode())
124             except:
125                 print(f"    Error enviando a hijo {ip_hijo}")
126                 self.eliminar_hijo(ip_hijo)
127
128     def enviar_mensaje(self, destino, mensaje):
129         id_mensaje = str(uuid.uuid4())
130         mensaje_completo = {
131             'id': id_mensaje,
132             'origen': self.ip,
133             'destino': destino,
134             'mensaje': mensaje,
135             'timestamp': time.time()
136         }
137
138         self.reenviar_mensaje_arbol(mensaje_completo, self.ip)
139
140     def eliminar_hijo(self, ip_hijo):
141         if ip_hijo in self.hijos:
142             del self.hijos[ip_hijo]
143             print(f"       Hijo desconectado: {ip_hijo}")
144
145     def procesar_mensaje_local(self, mensaje, origen=None):
146         if mensaje.startswith("ARDUINO:"):
147             comando = mensaje.split(":")[1].strip().upper()
148             if comando in ["ON", "OFF"]:
149                 if self.arduino:
150                     self.arduino.write(f"{comando}\n".encode())
151                     print(f"    Arduino ejecut  {comando}")
152                     if origen and origen != self.ip:
153                         self.enviar_mensaje(origen, f"ACK: Arduino {
                            comando} ejecutado")
154                 else:
155                     print(f"       Simulaci n Arduino {comando}")
156             else:
157                 print(f"    Comando Arduino inv lido: {comando}")
158         elif mensaje.startswith("REGISTRO:"):
159             pass
160         else:
161             print(f"       Mensaje local: {mensaje}")
162
163     def iniciar_servidor(self):
164         servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
165         servidor.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
166         servidor.bind((self.ip, self.puerto))
167         servidor.listen(5)
168
169         print(f"       {self.ip} ({self.rol}) escuchando en puerto {self
                .puerto}")
170
171         while True:
172             try:
173                 cliente, addr = servidor.accept()
174
```

```
175                 datos = cliente.recv(2048).decode()
176                 if datos:
177                     mensaje_registro = json.loads(datos)
178                     if mensaje_registro.get('tipo') == 'REGISTRO':
179                         self.aceptar_hijo(cliente, addr,
                                mensaje_registro)
180
181             except Exception as e:
182                 print(f"    Error aceptando conexi n: {e}")
183
184     def mostrar_estado(self):
185         print(f"\  n     ESTADO {self.ip} ({self.rol.upper()})")
186         print(f"       Padre: {self.padre.getpeername()[0] if self.padre
                else 'Ninguno'}")
187         print(f"       Hijos: {list(self.hijos.keys())}")
188         print(f"       Arduino: {'Conectado' if self.arduino else 'No
               disponible'}")
189
190 class Raiz(NodoArbol):
191     def __init__(self):
192         super().__init__('192.168.40.101', 'raiz')
193         self.nivel = 0
194
195     def conectar_arduino(self):
196         try:
197             self.arduino = serial.Serial('COM3', 9600, timeout=1)
198             time.sleep(2)
199             print("    Arduino conectado a la ra z")
200             return True
201         except:
202             print("    Arduino no disponible en la ra z")
203             return False
204
205     def iniciar(self):
206         print("=== NODO RA Z - TOPOLOG A  RBOL  ===")
207         self.conectar_arduino()
208
209         hilo_servidor = threading.Thread(target=self.iniciar_servidor,
               daemon=True)
210         hilo_servidor.start()
211
212         print("       Ra z iniciada - Esperando conexiones de ramas..."
               )
213         self.interfaz_usuario()
214
215     def interfaz_usuario(self):
216         while True:
217             entrada = input("RAIZ> ").strip()
218             if not entrada: continue
219             partes = entrada.split()
220             comando = partes[0].upper()
221
222             if comando == "SALIR": break
223             elif comando == "ESTADO": self.mostrar_estado()
224             elif comando == "ENVIAR" and len(partes) >= 3:
225                 destino = partes[1]
226                 mensaje = " ".join(partes[2:])
227                 self.enviar_mensaje(destino, mensaje)
```

```
            elif comando == "ARDUINO" and len(partes) >= 2:
                comando_arduino = " ".join(partes[1:])
                self.procesar_mensaje_local(f"ARDUINO:{comando_arduino}"
                    , self.ip)

if __name__ == "__main__":
    print("=" * 60)
    print("            TOPOLOG A   RBOL   - SELECCI N DE NODO")
    print("=" * 60)
    print("1.  Ra z  (192.168.40.101)")
    print("2.  Rama 1  (192.168.40.102)")
    print("3.  Rama 2  (192.168.40.103)")
    print("4.  Hoja 1  (192.168.40.104)")
    print("5.  Hoja 2  (192.168.40.105)")
    print("6.  Hoja 3  (192.168.40.106)")
    print("7.  Hoja 4  (192.168.40.107)")
    print("=" * 60)

    while True:
        opcion = input("Selecciona el n mero de tu nodo (1-7): ").strip
            ()
        if opcion == "1":
            nodo = Raiz()
            nodo.iniciar()
            break
        else:
            print("    Opci n no v lida")
```

Listing 4.1: Topología Árbol - Python

4.5. Código Fuente en Arduino

```
const int LED_PIN = 13;
int contador = 0;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
  delay(2000);
  Serial.println("ARDUINO ARBOL - COMANDOS: ON, OFF, STATUS");
}

void loop() {
  if (Serial.available() > 0) {
    String comando = Serial.readStringUntil('\n');
    comando.trim();
    comando.toUpperCase();

    Serial.print("Comando: ");
    Serial.println(comando);

    if (comando == "ON") {
      digitalWrite(LED_PIN, HIGH);
      Serial.println("LED ENCENDIDO");
    }
    else if (comando == "OFF") {
```

```
25        digitalWrite(LED_PIN, LOW);
26        Serial.println("LED APAGADO");
27      }
28      else if (comando == "STATUS") {
29        Serial.println("ARDUINO ACTIVO");
30      }
31      else {
32        Serial.println("COMANDO NO RECONOCIDO");
33      }
34    }
35    delay(100);
36  }
```

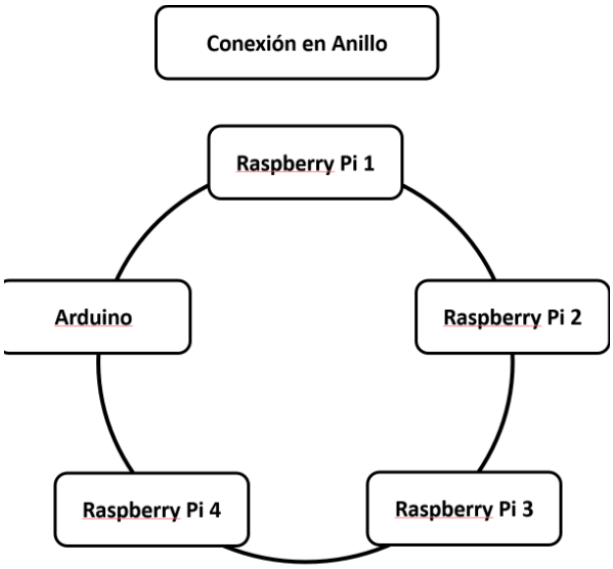
Listing 4.2: Topología Árbol - Arduino

Capítulo 5

Topología en Anillo

5.1. Descripción

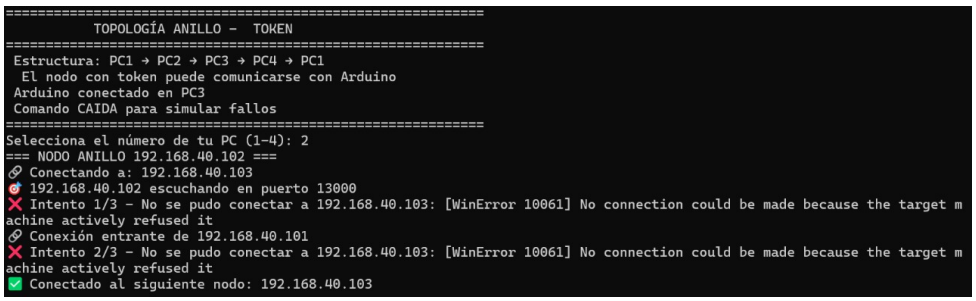
A continuación se presenta el código en Python del proyecto de topología en anillo con sistema de token:

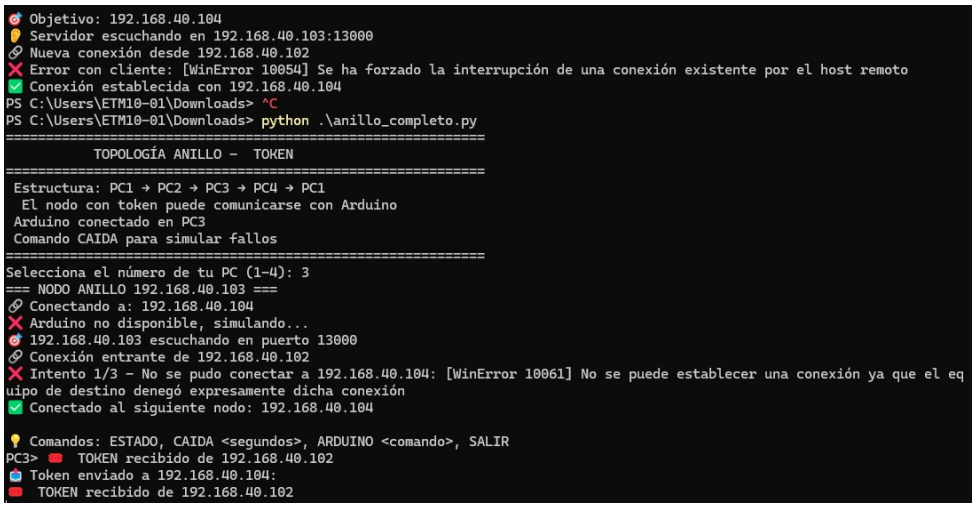


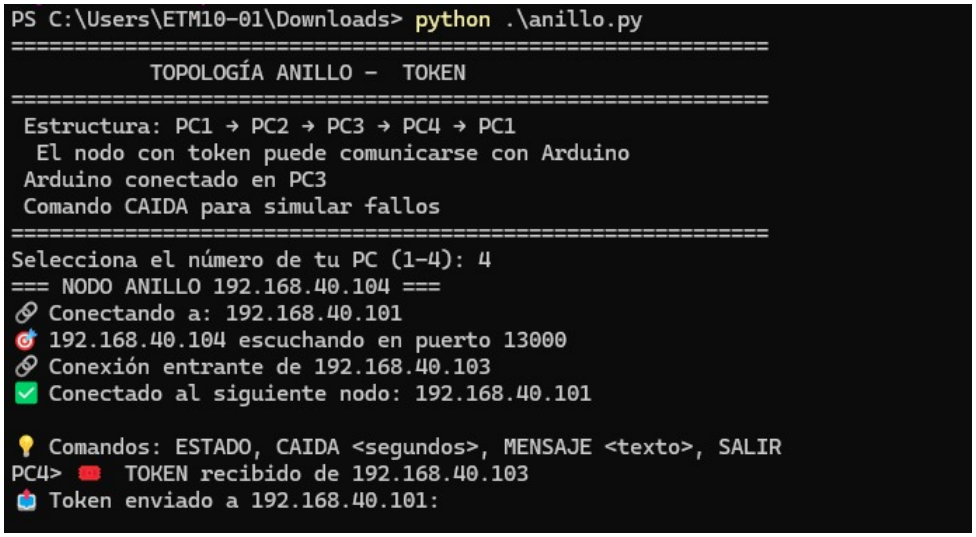
5.2. Resultados

```
KeyboardInterrupt
PS C:\Users\ETM10-01\Downloads> python .\anillo_completo.py
============================================================
          TOPOLOGÍA ANILLO -  TOKEN
============================================================
 Estructura: PC1 → PC2 → PC3 → PC4 → PC1
  El nodo con token puede comunicarse con Arduino
 Arduino conectado en PC3
 Comando CAIDA para simular fallos
============================================================
Selecciona el número de tu PC (1-4): 1
=== NODO ANILLO 192.168.40.101 ===
🔗 Conectando a: 192.168.40.102
🎯 192.168.40.101 escuchando en puerto 13000
✅ Conectado al siguiente nodo: 192.168.40.102
🚀 Nodo inicial - poseyendo token inicial
📩 Token enviado a 192.168.40.102: Token inicial del anillo

💡 Comandos: ESTADO, CAIDA <segundos>, MENSAJE <texto>, SALIR
PC1> 🔗 Conexión entrante de 192.168.40.104
🔴  TOKEN recibido de 192.168.40.104
```


TOPOLOGÍA ANILLO - TOKEN
Estructura: PC1 → PC2 → PC3 → PC4 → PC1
El nodo con token puede comunicarse con Arduino
Arduino conectado en PC3
Comando CAIDA para simular fallos
Selecciona el número de tu PC (1-4): 2
=== NODO ANILLO 192.168.40.102 ===
Conectando a: 192.168.40.103
192.168.40.102 escuchando en puerto 13000
Intento 1/3 - No se pudo conectar a 192.168.40.103: [WinError 10061] No connection could be made because the target machine actively refused it
Conexión entrante de 192.168.40.101
Intento 2/3 - No se pudo conectar a 192.168.40.103: [WinError 10061] No connection could be made because the target machine actively refused it
Conectado al siguiente nodo: 192.168.40.103


Objetivo: 192.168.40.104
Servidor escuchando en 192.168.40.103:13000
Nueva conexión desde 192.168.40.102
Error con cliente: [WinError 10054] Se ha forzado la interrupción de una conexión existente por el host remoto
Conexión establecida con 192.168.40.104
PS C:\Users\ETM10-01\Downloads> ^C
PS C:\Users\ETM10-01\Downloads> python .\anillo_completo.py
TOPOLOGÍA ANILLO - TOKEN
Estructura: PC1 → PC2 → PC3 → PC4 → PC1
El nodo con token puede comunicarse con Arduino
Arduino conectado en PC3
Comando CAIDA para simular fallos
Selecciona el número de tu PC (1-4): 3
=== NODO ANILLO 192.168.40.103 ===
Conectando a: 192.168.40.104
Arduino no disponible, simulando...
192.168.40.103 escuchando en puerto 13000
Conexión entrante de 192.168.40.102
Intento 1/3 - No se pudo conectar a 192.168.40.104: [WinError 10061] No se puede establecer una conexión ya que el equipo de destino denegó expresamente dicha conexión
Conectado al siguiente nodo: 192.168.40.104
Comandos: ESTADO, CAIDA <segundos>, ARDUINO <comando>, SALIR
PC3> TOKEN recibido de 192.168.40.102
Token enviado a 192.168.40.104:
TOKEN recibido de 192.168.40.102


PS C:\Users\ETM10-01\Downloads> python .\anillo.py
TOPOLOGÍA ANILLO - TOKEN
Estructura: PC1 → PC2 → PC3 → PC4 → PC1
El nodo con token puede comunicarse con Arduino
Arduino conectado en PC3
Comando CAIDA para simular fallos
Selecciona el número de tu PC (1-4): 4
=== NODO ANILLO 192.168.40.104 ===
Conectando a: 192.168.40.101
192.168.40.104 escuchando en puerto 13000
Conexión entrante de 192.168.40.103
Conectado al siguiente nodo: 192.168.40.101
Comandos: ESTADO, CAIDA <segundos>, MENSAJE <texto>, SALIR
PC4> TOKEN recibido de 192.168.40.103
Token enviado a 192.168.40.101:

5.3. Conexiones

5.4. Código Fuente en Python

```
import socket
import threading
import time
import json
import serial

# CLASES

class NodoAnillo:
    def __init__(self, ip_propia, ip_siguiente, tiene_arduino=False):
```

```
11          self.ip = ip_propia
12          self.ip_siguiente = ip_siguiente
13          self.tiene_arduino = tiene_arduino
14          self.puerto = 13000
15          self.token = False
16          self.arduino = None
17          self.conexion_activa = None
18          self.servidor_activo = True
19          self.nodo_siguiente_caido = False
20
21      def conectar_arduino(self):
22          if self.tiene_arduino:
23              try:
24                  self.arduino = serial.Serial('COM3', 9600, timeout=1)
25                  time.sleep(2)
26                  print(" Arduino conectado")
27                  return True
28              except:
29                  print(" Arduino no disponible, simulando...")
30                  return False
31
32      def conectar_siguiente_nodo(self):
33          """Conectar al siguiente nodo en el anillo"""
34          intentos = 0
35          while intentos < 3:
36              try:
37                  sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
38                  sock.settimeout(5)
39                  sock.connect((self.ip_siguiente, self.puerto))
40                  self.conexion_activa = sock
41                  self.nodo_siguiente_caido = False
42                  print(f" Conectado al siguiente nodo: {self.ip_siguiente
                        }")
43                  return True
44              except Exception as e:
45                  intentos += 1
46                  print(f" Intento {intentos}/3 - No se pudo conectar a {
                        self.ip_siguiente}: {e}")
47                  time.sleep(2)
48
49          self.nodo_siguiente_caido = True
50          print(f"    Nodo siguiente {self.ip_siguiente} considerado
                ca do")
51          return False
52
53      def enviar_token(self, mensaje_token=""):
54          """Enviar token al siguiente nodo"""
55          if not self.token:
56              print(" No tengo el token para enviar")
57              return False
58
59          if self.nodo_siguiente_caido:
60              print(f"    Nodo siguiente {self.ip_siguiente} est   ca do,
                    no se puede enviar token")
61              return False
62
63          try:
64              mensaje = {
```

```
                'tipo': 'TOKEN',
                'origen': self.ip,
                'destino': self.ip_siguiente,
                'mensaje': mensaje_token,
                'timestamp': time.time()
            }
            self.conexion_activa.send(json.dumps(mensaje).encode())
            self.token = False
            print(f" Token enviado a {self.ip_siguiente}: {mensaje_token}")
            return True
        except Exception as e:
            print(f" Error enviando token: {e}")
            self.nodo_siguiente_caido = True
            return False

    def manejar_conexion(self, cliente, addr):
        """Manejar conexiones entrantes (recibir token)"""
        try:
            while True:
                datos = cliente.recv(1024).decode()
                if not datos:
                    break

                mensaje = json.loads(datos)

                if mensaje['tipo'] == 'TOKEN':
                    print(f" TOKEN recibido de {mensaje['origen']}")
                    self.token = True

                    if mensaje['mensaje']:
                        print(f" Mensaje en token: {mensaje['mensaje']}")

                        if mensaje['mensaje'].startswith("ARDUINO:") and self.tiene_arduino:
                            comando = mensaje['mensaje'].split(":")[1]
                            self.procesar_comando_arduino(comando)

                    time.sleep(3)
                    self.enviar_token()

        except Exception as e:
            print(f" Error manejando conexi n: {e}")
        finally:
            cliente.close()

    def procesar_comando_arduino(self, comando):
        """Procesar comando para el Arduino"""
        print(f"    Ejecutando en Arduino: {comando}")

        if self.arduino:
            try:
                self.arduino.write(f"{comando}\n".encode())
                time.sleep(0.5)
                if self.arduino.in_waiting > 0:
                    respuesta = self.arduino.readline().decode().strip()
                    print(f" Arduino: {respuesta}")
```

```
120             except Exception as e:
121                 print(f" Error con Arduino: {e}")
122         else:
123             print(f" Simulaci n: Arduino ejecuta {comando}")
124 
125     def iniciar_servidor(self):
126         """Iniciar servidor para recibir token"""
127         servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
128         servidor.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
129         servidor.bind((self.ip, self.puerto))
130         servidor.listen(5)
131 
132         print(f" {self.ip} escuchando en puerto {self.puerto}")
133 
134         while self.servidor_activo:
135             try:
136                 cliente, addr = servidor.accept()
137                 print(f" Conexi n entrante de {addr[0]}")
138                 threading.Thread(target=self.manejar_conexion, args=(
                        cliente, addr), daemon=True).start()
139             except:
140                 break
141 
142         servidor.close()
143 
144     def simular_caida(self, duracion=10):
145         """Simular ca da de este nodo"""
146         print(f" Simulando ca da por {duracion} segundos...")
147         self.servidor_activo = False
148         if self.conexion_activa:
149             self.conexion_activa.close()
150 
151         time.sleep(duracion)
152 
153         print(" Reanudando operaci n...")
154         self.servidor_activo = True
155         self.conectar_siguiente_nodo()
156         threading.Thread(target=self.iniciar_servidor, daemon=True).
               start()
157 
158     def mostrar_estado(self):
159         """Mostrar estado del nodo"""
160         print(f"\n ESTADO ANILLO {self.ip}")
161         print(f" Token: {' S ' if self.token else 'NO'}")
162         print(f"    Siguiente nodo: {self.ip_siguiente}")
163         print(f" Estado siguiente nodo: {'ACTIVO' if not self.
               nodo_siguiente_caido else 'CA DO'}")
164         print(f" Arduino: {'CONECTADO' if self.tiene_arduino else 'NO'}"
               )
165 
166     def iniciar_anillo(self):
167         """Iniciar participaci n en el anillo"""
168         print(f"=== NODO ANILLO {self.ip} ===")
169         print(f"    Conectando a: {self.ip_siguiente}")
170 
171         if self.tiene_arduino:
172             self.conectar_arduino()
173 
```

```
174         threading.Thread(target=self.iniciar_servidor, daemon=True).
               start()
175         time.sleep(2)
176         self.conectar_siguiente_nodo()
177 
178         if self.ip == '192.168.40.101':
179             print("Nodo inicial - poseyendo token inicial")
180             time.sleep(5)
181             self.token = True
182             self.enviar_token("Token inicial del anillo")
183 
184         return True
185 
186 # CLASES PARA CADA NODO
187 
188 class PC1(NodoAnillo):
189     def __init__(self):
190         super().__init__('192.168.40.101', '192.168.40.102')
191 
192 class PC2(NodoAnillo):
193     def __init__(self):
194         super().__init__('192.168.40.102', '192.168.40.103')
195 
196 class PC3(NodoAnillo):
197     def __init__(self):
198         super().__init__('192.168.40.103', '192.168.40.104',
               tiene_arduino=True)
199 
200 class PC4(NodoAnillo):
201     def __init__(self):
202         super().__init__('192.168.40.104', '192.168.40.101')
203 
204 if __name__ == "__main__":
205     print("=" * 60)
206     print("           TOPOLOG A ANILLO - SISTEMA DE TOKEN")
207     print("=" * 60)
208     print("Estructura: PC1     PC2     PC3     PC4     PC1")
209     print("Solo el nodo con token puede comunicarse con Arduino")
210     print("Arduino conectado en PC3")
211     print("Comando CAIDA para simular fallos")
212     print("=" * 60)
213 
214     while True:
215         opcion = input("Selecciona el n mero de tu PC (1-4): ").strip()
216         if opcion == "1":
217             nodo = PC1()
218             break
219         elif opcion == "2":
220             nodo = PC2()
221             break
222         elif opcion == "3":
223             nodo = PC3()
224             break
225         elif opcion == "4":
226             nodo = PC4()
227             break
228         else:
229             print("Opci n no v lida. Usa 1, 2, 3 o 4")
```

```
230
231     if nodo.iniciar_anillo():
232         nodo.interfaz_usuario()
```

5.5. Código Fuente en Arduino

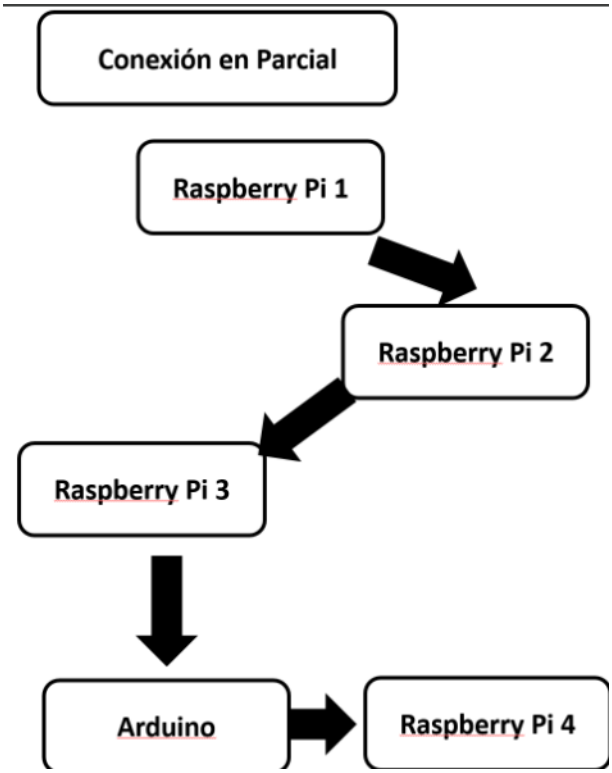
```
1  const int LED_PIN = 13;
2  int contadorComandos = 0;
3
4  void setup() {
5    pinMode(LED_PIN, OUTPUT);
6    digitalWrite(LED_PIN, LOW);
7    Serial.begin(9600);
8    delay(2000);
9    Serial.println("ARDUINO ANILLO - COMANDOS: ON, OFF, STATUS");
10 }
11
12 void loop() {
13   if (Serial.available() > 0) {
14     String comando = Serial.readStringUntil('\n');
15     comando.trim();
16     comando.toUpperCase();
17
18     contadorComandos++;
19
20     Serial.print("Comando #");
21     Serial.print(contadorComandos);
22     Serial.print(": ");
23     Serial.println(comando);
24
25     if (comando == "ON") {
26       digitalWrite(LED_PIN, HIGH);
27       Serial.println("LED ENCENDIDO - Token funcionando");
28     }
29     else if (comando == "OFF") {
30       digitalWrite(LED_PIN, LOW);
31       Serial.println("LED APAGADO - Token funcionando");
32     }
33     else if (comando == "STATUS") {
34       Serial.println("ARDUINO ACTIVO EN TOPOLOGIA ANILLO");
35       Serial.print("Total comandos recibidos: ");
36       Serial.println(contadorComandos);
37     }
38     else {
39       Serial.println("COMANDO NO RECONOCIDO - Usar: ON, OFF, STATUS");
40     }
41   }
42   delay(100);
43 }
```

Capítulo 6

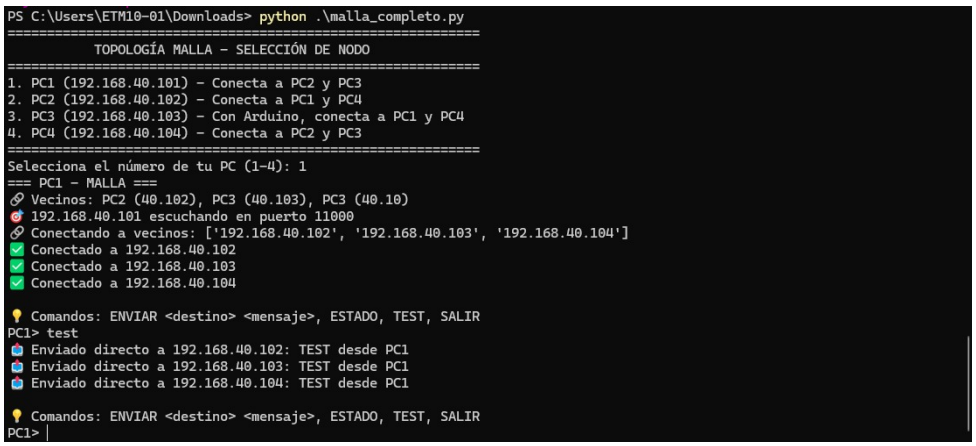
Topología en Malla Parcial

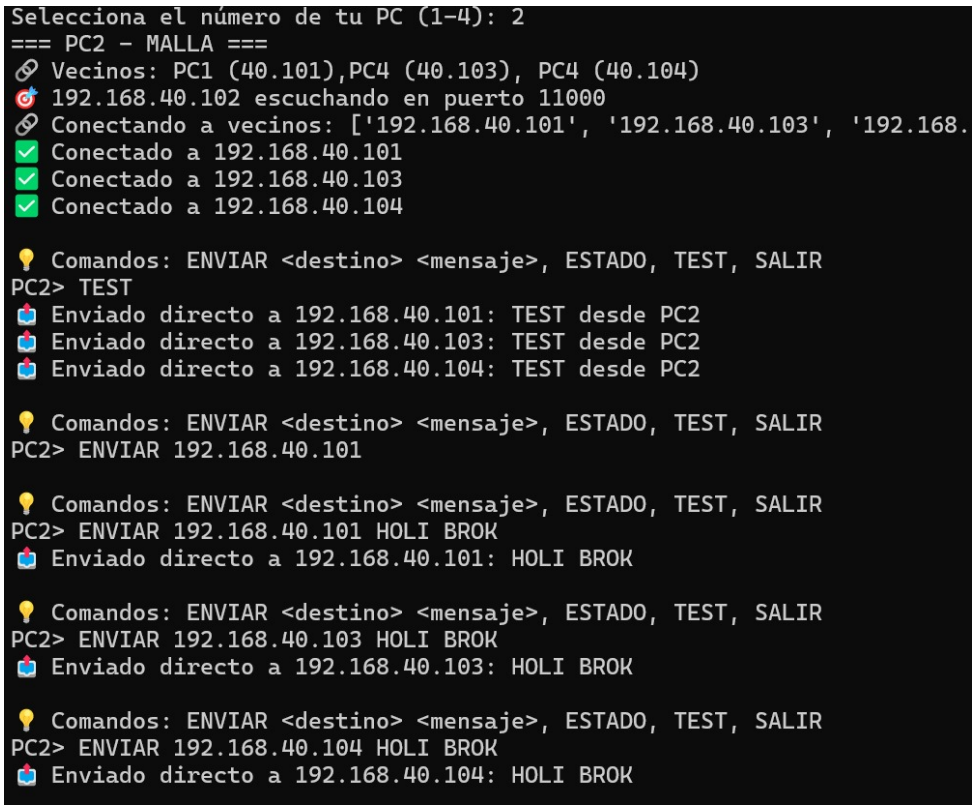
6.1. Descripción

A continuación se presenta el código en Python del proyecto de topología en parcial con sistema de token:



6.2. Resultados





```
1. PC1 (192.168.40.101) - Conecta a PC2 y PC3
2. PC2 (192.168.40.102) - Conecta a PC1 y PC4
3. PC3 (192.168.40.103) - Con Arduino, conecta a PC1 y PC4
4. PC4 (192.168.40.104) - Conecta a PC2 y PC3
============================================================
Selecciona el número de tu PC (1-4): 3
=== PC3 - MALLA (CON ARDUINO) ===
🔗 Vecinos: PC1 (40.101), PC4 (40.104)
❌ Arduino no disponible, simulando...
🎯 192.168.40.103 escuchando en puerto 11000
🔗 Conectando a vecinos: ['192.168.40.101', '192.168.40.102', '192.168.40.104']
✅ Conectado a 192.168.40.101
✅ Conectado a 192.168.40.102
✅ Conectado a 192.168.40.104

💡 Comandos: ENVIAR <destino> <mensaje>, ARDUINO <cmd>, ESTADO, SALIR
PC3> ENVIAR 192.168.40.102 :)
📤 Enviado directo a 192.168.40.102: :)

💡 Comandos: ENVIAR <destino> <mensaje>, ARDUINO <cmd>, ESTADO, SALIR
PC3> ENVIAR 192.168.40.101 :)
📤 Enviado directo a 192.168.40.101: :)

💡 Comandos: ENVIAR <destino> <mensaje>, ARDUINO <cmd>, ESTADO, SALIR
PC3> ENVIAR 192.168.40.104 :)
📤 Enviado directo a 192.168.40.104: :)

💡 Comandos: ENVIAR <destino> <mensaje>, ARDUINO <cmd>, ESTADO, SALIR
PC3> 🔗 Nueva conexión de 192.168.40.101
```

```
PS C:\Users\ETM10-01\Downloads> python .\malla_completo.py
============================================================
           TOPOLOGÍA MALLA - SELECCIÓN DE NODO
============================================================
1. PC1 (192.168.40.101) - Conecta a PC2 y PC3
2. PC2 (192.168.40.102) - Conecta a PC1 y PC4
3. PC3 (192.168.40.103) - Con Arduino, conecta a PC1 y PC4
4. PC4 (192.168.40.104) - Conecta a PC2 y PC3
============================================================
Selecciona el número de tu PC (1-4): 4
=== PC4 - MALLA ===
🔗 Vecinos: PC2 (40.102), PC3 (40.103),PC3 (40.101)
🎯 192.168.40.104 escuchando en puerto 11000
🔗 Conectando a vecinos: ['192.168.40.102', '192.168.40.103', '192.168.40.101']
✅ Conectado a 192.168.40.102
✅ Conectado a 192.168.40.103
✅ Conectado a 192.168.40.101

💡 Comandos: ENVIAR <destino> <mensaje>, ESTADO, TEST, SALIR
PC4> TEST
📤 Enviado directo a 192.168.40.101: TEST desde PC4
📤 Enviado directo a 192.168.40.102: TEST desde PC4
📤 Enviado directo a 192.168.40.103: TEST desde PC4
```

6.3. Conexiones

6.4. Código Fuente en Python

```
import socket
import threading
import time
import json
import serial
import uuid


class NodoMalla:
    def __init__(self, ip_propia, vecinos):
```

```
        self.ip = ip_propia
        self.vecinos = vecinos
        self.nodos_conectados = {}
        self.nodos_caidos = set()
        self.puerto = 11000
        self.arduino = None
        self.mensajes_recibidos = set()  # evitar loops

    def conectar_vecino(self, ip_vecino):
        """Conectar a un vecino de la malla"""
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            sock.settimeout(5)
            sock.connect((ip_vecino, self.puerto))
            self.nodos_conectados[ip_vecino] = sock
            print(f"Conectado a {ip_vecino}")
            return sock
        except Exception as e:
            print(f"No se pudo conectar a {ip_vecino}: {e}")
            self.nodos_caidos.add(ip_vecino)
            return None

    def enviar_mensaje(self, destino, mensaje, id_mensaje=None):
        """Enviar mensaje con ID  nico  para flooding"""
        if id_mensaje is None:
            id_mensaje = str(uuid.uuid4())

        mensaje_completo = json.dumps({
            'id': id_mensaje,
            'origen': self.ip,
            'destino': destino,
            'mensaje': mensaje,
            'timestamp': time.time()
        })

        # enviar a TODOS los vecinos
        for vecino, sock in self.nodos_conectados.items():
            try:
                sock.send(mensaje_completo.encode())
            except:
                print(f"Error enviando a {vecino}")
                self.nodos_caidos.add(vecino)

    def manejar_conexion(self, cliente, addr):
        """Manejar mensajes entrantes"""
        try:
            while True:
                datos = cliente.recv(2048).decode()
                if not datos:
                    break

                try:
                    mensaje = json.loads(datos)
                except:
                    continue

                # evitar procesar el mismo mensaje dos veces
                if mensaje['id'] in self.mensajes_recibidos:
```

```
                    continue
                self.mensajes_recibidos.add(mensaje['id'])

                print(f"[{self.ip}] De {mensaje['origen']} para {mensaje['destino']}: {mensaje['mensaje']}")

                # Si el mensaje es para este nodo
                if mensaje['destino'] == self.ip:
                    self.procesar_mensaje_local(mensaje['mensaje'], mensaje['origen'])
                else:
                    # reenviar (flooding) excepto al origen inmediato
                    self.enviar_mensaje(mensaje['destino'], mensaje['mensaje'], id_mensaje=mensaje['id'])

        except Exception as e:
            print(f"Error manejando conexi n: {e}")
        finally:
            cliente.close()

    def procesar_mensaje_local(self, mensaje, origen=None):
        """Procesar mensaje destinado a este nodo"""
        if mensaje.startswith("ARDUINO:"):
            comando = mensaje.split(":")[1]
            print(f"Comando Arduino recibido: {comando}")
        else:
            print(f"Mensaje local: {mensaje}")

    def iniciar_servidor(self):
        """Iniciar servidor para recibir conexiones"""
        servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        servidor.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        servidor.bind((self.ip, self.puerto))
        servidor.listen(5)

        print(f"{self.ip} escuchando en puerto {self.puerto}")

        while True:
            try:
                cliente, addr = servidor.accept()
                threading.Thread(target=self.manejar_conexion, args=(cliente, addr)).start()
            except Exception as e:
                print(f"Error aceptando conexi n: {e}")

    def conectar_malla(self):
        """Conectar con todos los vecinos"""
        for vecino in self.vecinos:
            if vecino not in self.nodos_caidos:
                self.conectar_vecino(vecino)

    def mostrar_estado(self):
        """Mostrar estado de la malla"""
        print(f"\nESTADO {self.ip}")
        print(f"Conectados: {list(self.nodos_conectados.keys())}")
        print(f"Ca dos: {list(self.nodos_caidos)}")
        print(f"Vecinos: {self.vecinos}")

```

```
123
124 class PC1(NodoMalla):
125     def __init__(self):
126         super().__init__('192.168.40.101', ['192.168.40.102', '
                192.168.40.103', '192.168.40.104'])
127
128     def iniciar(self):
129         print("=== PC1 - MALLA ===")
130         hilo_servidor = threading.Thread(target=self.iniciar_servidor,
                daemon=True)
131         hilo_servidor.start()
132         time.sleep(2)
133         self.conectar_malla()
134         self.interfaz_usuario()
135
136     def interfaz_usuario(self):
137         while True:
138             entrada = input("PC1> ").strip()
139             if not entrada: continue
140             partes = entrada.split()
141             comando = partes[0].upper()
142
143             if comando == "SALIR": break
144             elif comando == "ESTADO": self.mostrar_estado()
145             elif comando == "ENVIAR" and len(partes) >= 3:
146                 destino = partes[1]
147                 mensaje = " ".join(partes[2:])
148                 self.enviar_mensaje(destino, mensaje)
149
150
151 class PC2(NodoMalla):
152     def __init__(self):
153         super().__init__('192.168.40.102', ['192.168.40.101', '
                192.168.40.104'])
154
155     def iniciar(self):
156         print("=== PC2 - MALLA ===")
157         hilo_servidor = threading.Thread(target=self.iniciar_servidor,
                daemon=True)
158         hilo_servidor.start()
159         time.sleep(2)
160         self.conectar_malla()
161         self.interfaz_usuario()
162
163     def interfaz_usuario(self):
164         while True:
165             entrada = input("PC2> ").strip()
166             if not entrada: continue
167             partes = entrada.split()
168             comando = partes[0].upper()
169
170             if comando == "SALIR": break
171             elif comando == "ESTADO": self.mostrar_estado()
172             elif comando == "ENVIAR" and len(partes) >= 3:
173                 destino = partes[1]
174                 mensaje = " ".join(partes[2:])
175                 self.enviar_mensaje(destino, mensaje)
176
```

```
177
178 class PC3(NodoMalla):
179     def __init__(self):
180         super().__init__('192.168.40.103', ['192.168.40.101', '
                192.168.40.104'])
181         self.arduino = None
182
183     def conectar_arduino(self):
184         try:
185             self.arduino = serial.Serial('COM3', 9600, timeout=1)
186             time.sleep(2)
187             print("Arduino conectado en PC3")
188             return True
189         except:
190             print("Arduino no disponible, simulando...")
191             return False
192
193     def procesar_mensaje_local(self, mensaje, origen=None):
194         if mensaje.startswith("ARDUINO:"):
195             comando = mensaje.split(":")[1].strip().upper()
196             if comando in ["ON", "OFF"]:
197                 if self.arduino:
198                     self.arduino.write(f"{comando}\n".encode())
199                     print(f"Arduino ejecut  {comando}")
200                 else:
201                     print(f"Simulaci n Arduino {comando}")
202             else:
203                 print(f"Comando Arduino inv lido: {comando}")
204         else:
205             print(f"Mensaje: {mensaje}")
206
207     def iniciar(self):
208         print("=== PC3 - MALLA (CON ARDUINO) ===")
209         self.conectar_arduino()
210         hilo_servidor = threading.Thread(target=self.iniciar_servidor,
                daemon=True)
211         hilo_servidor.start()
212         time.sleep(2)
213         self.conectar_malla()
214         self.interfaz_usuario()
215
216     def interfaz_usuario(self):
217         while True:
218             entrada = input("PC3> ").strip()
219             if not entrada: continue
220             partes = entrada.split()
221             comando = partes[0].upper()
222
223             if comando == "SALIR": break
224             elif comando == "ESTADO": self.mostrar_estado()
225
226
227 class PC4(NodoMalla):
228     def __init__(self):
229         super().__init__('192.168.40.104', ['192.168.40.102', '
                192.168.40.103'])
230
231     def iniciar(self):
```

```
        print("=== PC4 - MALLA ===")
        hilo_servidor = threading.Thread(target=self.iniciar_servidor,
            daemon=True)
        hilo_servidor.start()
        time.sleep(2)
        self.conectar_malla()
        self.interfaz_usuario()

    def interfaz_usuario(self):
        while True:
            entrada = input("PC4> ").strip()
            if not entrada: continue
            partes = entrada.split()
            comando = partes[0].upper()

            if comando == "SALIR": break
            elif comando == "ESTADO": self.mostrar_estado()
            elif comando == "ENVIAR" and len(partes) >= 3:
                destino = partes[1]
                mensaje = " ".join(partes[2:])
                self.enviar_mensaje(destino, mensaje)

if __name__ == "__main__":
    print("=" * 60)
    print("            TOPOLOG A MALLA - SELECCI N DE NODO")
    print("=" * 60)
    print("1. PC1 (192.168.40.101)")
    print("2. PC2 (192.168.40.102)")
    print("3. PC3 (192.168.40.103) - Arduino")
    print("4. PC4 (192.168.40.104)")
    print("=" * 60)

    while True:
        opcion = input("Selecciona el n mero de tu PC (1-4): ").strip()
        if opcion == "1":
            nodo = PC1()
            break
        elif opcion == "2":
            nodo = PC2()
            break
        elif opcion == "3":
            nodo = PC3()
            break
        elif opcion == "4":
            nodo = PC4()
            break
        else:
            print(" Opci n no v lida")

    nodo.iniciar()
```

6.5. Código Fuente en Arduino

```
const int LED_PIN = 13;
int contador = 0;

```

```
4  void setup() {
5    pinMode(LED_PIN, OUTPUT);
6    Serial.begin(9600);
7    delay(2000);
8    Serial.println("ARDUINO MALLA - COMANDOS: ON, OFF, STATUS");
9  }
10
11 void loop() {
12   if (Serial.available() > 0) {
13     String comando = Serial.readStringUntil('\n');
14     comando.trim();
15     comando.toUpperCase();
16
17     Serial.print("Comando: ");
18     Serial.println(comando);
19
20     if (comando == "ON") {
21       digitalWrite(LED_PIN, HIGH);
22       Serial.println("LED ENCENDIDO");
23     }
24     else if (comando == "OFF") {
25       digitalWrite(LED_PIN, LOW);
26       Serial.println("LED APAGADO");
27     }
28     else if (comando == "STATUS") {
29       Serial.println("ARDUINO ACTIVO");
30     }
31     else {
32       Serial.println("COMANDO NO RECONOCIDO");
33     }
34   }
35   delay(100);
36 }
```

Capítulo 7

Conclusión

En este laboratorio se implementaron y configuraron diferentes topologías de red: estrella, árbol, anillo y malla parcial. Cada topología presenta características únicas en términos de:

- **Estrella:** Centralizada, fácil administración pero punto único de fallo
- **Árbol:** Jerárquica, escalable y organizada
- **Anillo:** Sistema de token, acceso ordenado pero susceptible a fallos en cadena
- **Malla Parcial:** Redundante, múltiples caminos pero mayor complejidad

Todas las implementaciones incluyeron comunicación con Arduino para control remoto, demostrando la integración entre redes de computadoras y sistemas embebidos.