Tarea 1

CC5326 – Diseño de Internet de las Cosas

Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Link: `https://iiot-embedded.com/iot/tareas/t1/`

Profesor: Luciano Radrigán F.
Ayudante: Gabriel Díaz C.

Objetivos a cumplir

- Generar una red WiFi a través de la Raspberry Pi (con nombre y contraseña), que otorgue IPs a los ESP32 mediante DHCP.
- Programar un servidor de socket TCP y UDP en la Raspberry Pi.
- Programar un cliente de socket TCP y UDP en el ESP32.
- Enviar un paquete desde el ESP32 con datos generados en el mismo dispositivo, que será recibido por el servidor.
- Almacenar la información en una base de datos PostgreSQL utilizando Docker.
- Posibilitar el cambio del protocolo de envío de datos desde la base de datos.
- Permitir el cambio del tipo de conexión (TCP o UDP) desde la base de datos.
- Los cambios en la base de datos deberán reflejarse en el ESP32.
- Recibir datos simultáneamente de dos ESP32 y almacenarlos en la base de datos.

Indicaciones

Flujo de los datos

1. El ESP32 se inicia mediante conexión TCP.
2. Consulta la base de datos por `ID_protocol` y `Transport_Layer`.

3. Dependiendo del tipo de conexión:
    - **TCP:** Envío de paquete, luego Deep Sleep por 60 segundos.
    - **UDP:** Envío continuo hasta cambiar valor de `Transport_Layer`.
4. La Raspberry Pi debe descomponer los paquetes y almacenarlos en la base de datos SQL.
5. En cada envío se debe verificar configuración actual.

Encabezados de los paquetes

- **Header (2 bytes)**
- **ID** (6 bytes)
- **Device MAC** (1 byte)
- **Transport Layer** (1 byte)
- **ID Protocol** (2 bytes)
- **Length** (2 bytes)

Protocolos de envío

- Cada protocolo define los datos a enviar según configuración en la base de datos.
- Todos siguen el formato `Headers + Body`.

Ejemplo de datos por protocolo

Protocolos 0 al 3

- Data 1: Batt level (1 byte)
- Data 2: Timestamp (4 bytes)
- Data 3: Temp (1 byte)
- Data 4: Press (4 bytes)
- Data 5: Hum (1 byte)
- Data 6: CO (4 bytes)
- Data 7: RMS (4 bytes)
- Data 8–13: Amp/Frec en X, Y, Z (4 bytes cada uno)

Protocolo 4

- Igual que anterior hasta Data 6.
- Data 7–12: Vectores de 2000 valores float (8000 bytes cada uno)

Generación de datos

Acceloremeter_kpi

- Ampx: 0.0059 – 0.12
- Freqx: 29.0 – 31.0
- Apy: 0.0041 – 0.11
- Freqy: 59.0 – 61.0
- Ampz: 0.008 – 0.15
- Freqz: 89.0 – 91.0
- RMS: $\sqrt{Ampx^2 + Ampy^2 + Ampz^2}$

Acceloremeter_Sensor

- Acc_X/Y/Z: -16.0 – 16.0
- Rgyr_X/Y/Z: -1000 – 1000

THPC_Sensor

- Temp: 5 – 30
- Hum: 30 – 80
- Pres: 1000 – 1200
- CO: 30 – 200 (float)

Batt_Sensor

- Nivel batería: uint8 entre 1 y 100

Base de Datos

Tablas requeridas

- **Datos:** Guarda todos los datos con timestamp y dispositivo.
- **Logs:** Guarda conexiones con ID, capa de transporte, protocolo y timestamp.
- **Configuración:** Contiene `ID_protocol` y `Transport_Layer`.
- **Loss:** Tiempo de retardo y pérdida de paquetes (en bytes).

Inicialización con Docker

```
docker-compose up -d
```

Entrega

- Demostración en vivo el lunes 28 de abril.
- Subir código a U-Cursos (o enlace a repositorio Git).

Recomendaciones

- Coordinar pruebas físicas en persona.
- Usar repositorio compartido (como GitHub).
- Conexiones físicas: revisar pines antes de usar.
- Verificar componentes antes de culpar al software.
- Consultas: foro o equipo docente.