Documentación de Funciones

`Nodo.cpp` y `Modem.ino`

Autores: ...

Índice

1 Funciones implementadas en `Nodo.cpp`

1.1 Constructor y Destructor

`Nodo(uint16_t ip)` Constructor que inicializa el nodo con una dirección IP específica. También inicializa el contador de ID de mensajes y configura estructuras de almacenamiento para los nodos conocidos y los ACKs pendientes.

`~Nodo()` Destructor que se encarga de restaurar configuraciones (como entrada no bloqueante) y liberar cualquier recurso utilizado por el nodo.

1.2 Función principal

`void run()` Es la función que arranca el ciclo principal del nodo. Se encarga de mostrar los menús al usuario, actualizar los mensajes entrantes, verificar ACKs y procesar la interacción completa con el módem a través de UART.

1.3 Menús

`void menu()` Muestra el menú principal con las opciones disponibles para el usuario. Permite acceder al envío de mensajes, ver nodos, y entrar al submenú de comandos internos.

`void menuComandosInternos()` Muestra las opciones de comandos del protocolo propio (como LED, OLED o prueba). Estos comandos son enviados directamente al módem como protocolo interno.

`void menuEnvioMensajes()` Permite seleccionar el tipo de mensaje a enviar: Unicast, Broadcast, Hello u otros comandos específicos.

1.4 Comunicación

`void actualizarMensajesEntrantes()` Lee continuamente los datos desde UART, desempaqueta los mensajes SLIP y luego los interpreta como paquetes IPv4 simplificados. Llama a las funciones de procesamiento adecuadas según el tipo de mensaje.

`void enviarPaquete(const IPv4 &paquete)` Empaqueta un paquete IPv4 en SLIP y lo envía por UART al módem.

`void enviarACK(uint16_t ip_destino, uint16_t id_mensaje)` Crea un paquete IPv4 con el protocolo 1 (ACK) para confirmar la recepción de un mensaje y lo envía al nodo de origen.

`void enviarComandoAlModem(const PropioProtocolo &comando)` Envia directamente un mensaje tipo protocolo propio (protocolo 0) al módem, utilizado para comandos internos como cambiar LED, mostrar mensaje en OLED, etc.

`void verificarACKsPendientes()` Revisa si los ACKs pendientes han sido recibidos. Si no, intenta reenviar hasta dos veces el mensaje original antes de descartarlo e informar al usuario.

1.5 Procesamiento de mensajes recibidos

`void procesarACK(const IPv4 &paquete)` Al recibir un ACK, elimina el mensaje correspondiente de la lista de pendientes.

`void procesarMensajeUnicast(const IPv4 &paquete)` Muestra en pantalla el contenido del mensaje unicast recibido y responde con un ACK.

`void procesarMensajeBroadcast(const IPv4 &paquete)` Similar al unicast, pero sin necesidad de responder con ACK.

`void procesarHello(const IPv4 &paquete)` Registra la IP del nodo que envió el mensaje hello y el tiempo en que fue recibido.

`void procesarComandoPrueba(const IPv4 &paquete)` Prepara y envía un comando para que el módem muestre una imagen o texto de prueba.

`void procesarComandoLed(const IPv4 &paquete)` Envía una señal al módem para alternar el estado del LED interno.

`void procesarComandoOLED(const IPv4 &paquete)` Toma el mensaje del paquete y lo convierte en comando para que el módem lo muestre en la pantalla OLED.

1.6 Envío de mensajes

`void verNodos()` Muestra por pantalla una tabla con las IPs de los nodos que han enviado mensajes Hello y el tiempo desde su recepción.

`void enviarHello()` Envía un mensaje broadcast con el contenido "hola"para que otros nodos lo registren.

`void enviarMensajeUnicast()` Permite al usuario escribir un mensaje y la IP destino. Lo envía como protocolo 2, y espera un ACK como confirmación.

`void enviarMensajeBroadcast()` Similar al unicast, pero el mensaje se dirige a todos los nodos.

`void enviarComandoPrueba()` Envia un paquete con protocolo 5 (comando vacío) al módem destino para pruebas.

`void enviarComandoLed()` Envia un comando tipo protocolo 6 para alternar el LED del receptor.

`void enviarMensajeOLED()` Permite ingresar un mensaje de texto para ser mostrado en la pantalla OLED del receptor (protocolo 7).

1.7 Utilidades

`uint16_t obtenerNuevoID()` Genera un nuevo ID de mensaje incremental que se usará en los encabezados IPv4.

`void limpiarPantalla()` Limpia la consola de texto en la terminal.

1.8 Manejo de entrada no bloqueante

`void configurarEntradaNoBloqueante()` Configura la entrada estándar para no bloquear la ejecución del programa cuando no hay input del usuario.

`void restaurarEntradaOriginal()` Restaura la configuración original de la entrada estándar.

`bool leerLineaNoBloqueante(std::string &buffer)` Intenta leer una línea desde la entrada sin detener el flujo del programa. Retorna verdadero si se leyó algo.

2 Explicación detallada de las funciones en `Modem.ino`

2.1 setup()

```
void setup()
```

Esta función se ejecuta una vez al inicio del programa. Su propósito principal es:

- Inicializar la comunicación serial con la Raspberry Pi (UART).
- Configurar los pines del LED u otros periféricos como salidas.
- Inicializar la pantalla OLED si es utilizada.
- Configurar el módulo LoRa con sus parámetros (frecuencia, ancho de banda, potencia, etc.).

También puede establecer la dirección IP del módem y mostrar un mensaje inicial en la OLED para fines de depuración o identificación.

2.2 loop()

```
void loop()
```

Función principal del ciclo infinito de la ESP32. En ella se realiza:

- Lectura de datos entrantes por UART (desde la Raspberry).
- Desempaquetado del mensaje SLIP.
- Procesamiento del paquete IPv4 simplificado.
- Reenvío por LoRa si el paquete no es para este nodo.
- Ejecución de comandos si el paquete es protocolo propio y destinado al módem.
- Envío de mensajes recibidos por LoRa a través de UART a la Raspberry si el destino es este nodo o es broadcast.

2.3 procesarMensajeIP(const IPv4 &paquete)

```
void procesarMensajeIP(const IPv4 &paquete)
```

Función interna que determina la acción a tomar con un paquete recibido por UART:

- Si es protocolo 0 y el destino es la IP local: ejecuta el comando.
- Si es protocolo distinto y no es para este nodo: lo reenvía por LoRa.
- Si no cumple lo anterior, el paquete es descartado.

2.4 enviarPorUART(const IPv4 &paquete)

```
void enviarPorUART(const IPv4 &paquete)
```

Encapsula un paquete IPv4 simplificado en SLIP y lo transmite por UART a la Raspberry. Se utiliza para enviar los paquetes recibidos por LoRa al nodo correspondiente.

2.5 enviarPorLoRa(const IPv4 &paquete)

```
void enviarPorLoRa(const IPv4 &paquete)
```

Convierte el paquete IPv4 en un arreglo de bytes y lo envía usando la radio LoRa. Usado para retransmitir mensajes no dirigidos directamente al módem.

2.6 procesarComandoPropio(const PropioProtocolo &cmd)

```
void procesarComandoPropio(const PropioProtocolo &cmd)
```

Desempaqueta el comando del protocolo propio y actúa dependiendo del tipo:

- Comando 5: Muestra un mensaje de prueba en la pantalla OLED.
- Comando 6: Alterna el estado del LED.
- Comando 7: Muestra texto recibido en la pantalla OLED.

2.7 leerDeUART() y leerDeLoRa()

Estas funciones (usualmente con nombres como `leerDeUART()` o `leerDeLoRa()`) se encargan de:

- Detectar si hay datos disponibles.
- Leer y almacenar los bytes entrantes.
- Reconstruir el paquete (si aplica SLIP o estructura fija).
- Validar IP de destino, protocolos y decidir el flujo.

2.8 Funciones auxiliares

También es común que el archivo contenga funciones adicionales:

- `configurarLoRa()`: inicializa el módulo LoRa con parámetros apropiados.
- `mostrarEnOLED(String msg)`: imprime un mensaje en pantalla.
- `toggleLED()`: cambia el estado del LED.
- `paqueteEsParaMi(const IPv4 &p)`: determina si un paquete debe ser procesado localmente.